DSTni-EX User Guide

Section Six

Part Number 900-335
Revision A 3/04

Copyright & Trademark

Lantronix
15353 Barranca Parkway
Irvine, CA 92618, USA
Phone: 949-453-3990
Fax: 949-453-3995

Technical Support
Phone: 630-245-1445
Fax: 630-245-1717

Master Distributor
Grid Connect
1841 Centre Point Circle, Suite 143
Naperville, IL 60563
Phone: 630-245-1445
www.gridconnect.com

REV	Changes	Released Date
A	Reformat. Add changes from Design Spec. 1.1	3-24-04

Warranty

Lantronix warrants each Lantronix product to be free from defects in material and workmanship for a period specified on the product warranty registration card after the date of shipment. During this period, if a customer is unable to resolve a product problem with Lantronix Technical Support, a Return Material Authorization (RMA) will be issued. Following receipt of an RMA number, the customer shall return the product to Lantronix, freight prepaid. Upon verification of warranty, Lantronix will -- at its option -- repair or replace the product and return it to the customer freight prepaid. If the product is not under warranty, the customer may have Lantronix repair the unit on a fee basis or return it. No services are handled at the customer's site under this warranty. This warranty is voided if the customer uses the product in an unauthorized or improper way, or in an environment for which it was not designed.

Lantronix warrants the media containing its software product to be free from defects and warrants that the software will operate substantially according to Lantronix specifications for a period of **60 DAYS** after the date of shipment. The customer will ship defective media to Lantronix. Lantronix will ship the replacement media to the customer.

* * * *

In no event will Lantronix be responsible to the user in contract, in tort (including negligence), strict liability or otherwise for any special, indirect, incidental or consequential damage or loss of equipment, plant or power system, cost of capital, loss of profits or revenues, cost of replacement power, additional expenses in the use of existing software, hardware, equipment or facilities, or claims against the user by its employees or customers resulting from the use of the information, recommendations, descriptions and safety notations supplied by Lantronix. Lantronix liability is limited (at its election) to:

refund of buyer's purchase price for such affected products (without interest)

repair or replacement of such products, provided that the buyer follows the above procedures.

There are no understandings, agreements, representations or warranties, express or implied, including warranties of merchantability or fitness for a particular purpose, other than those specifically set out above or by any existing contract between the parties. Any such contract states the entire obligation of Lantronix. The contents of this document shall not become part of or modify any prior or existing agreement, commitment or relationship.

For details on the Lantronix warranty replacement policy, go to our web site at http://www.lantronix.com/support/warranty/index.html

Contents

List of Figures

List of Tables

1: About This User Guide

This User Guide describes the technical features and programming interfaces of the Lantronix DSTni-EX chip (hereafter referred to as “DSTni”).

DSTni is an Application Specific Integrated Circuit (ASIC)-based single-chip solution (SCS) that integrates the leading-edge functionalities needed to develop low-cost, high-performance device server products. On a single chip, the DSTni integrates an x186 microprocessor, 16K-byte ROM, 256K-byte SRAM, programmable input/output (I/O), and serial, Ethernet, and Universal Serial Bus (USB) connectivity — key ingredients for device- server solutions. Although DSTni embeds multiple functions onto a single chip, it can be easily customized, based on the comprehensive feature set designed into the chip.

Providing a complete device server solution on a single chip enables system designers to build affordable, full-function solutions that provide the highest level of performance in both processing power and peripheral systems, while reducing the number of total system components. The advantages gained from this synergy include:

- Simplifying system design and increased reliability.
- Minimizing marketing and administration costs by eliminating the need to source products from multiple vendors.
- Eliminating the compatibility and reliability problems that occur when combining separate subsystems.
- Dramatically reducing implementation costs.
- Increasing performance and functionality, while maintaining quality and cost effectiveness.
- Streamlining development by reducing programming effort and debugging time.
- Enabling solution providers to bring their products to market faster.

These advantages make DSTni the ideal solution for designs requiring x86 compatibility; increased performance; serial, programmable I/O, Ethernet, and USB communications; and a glueless bus interface.

Intended Audience

This User Guide is intended for use by hardware and software engineers, programmers, and designers who understand the basic operating principles of microprocessors and their systems and are considering designing systems that utilize DSTni.

Conventions

This User Guide uses the following conventions to alert you to information of special interest.

The symbols # and n are used throughout this Guide to denote active LOW signals.

Notes: *Notes are information requiring attention.*

Navigating Online

The electronic Portable Document Format (PDF) version of this User Guide contains *hyperlinks*. Clicking one of these hyper links moves you to that location in this User Guide. The PDF file was created with Bookmarks and active links for the Table of Contents, Tables, Figures and cross-references.

Organization

This User Guide contains information essential for system architects and design engineers. The information in this User Guide is organized into the following chapters and appendixes.

- *Section 1: Introduction*
  Describes the DSTni architecture, design benefits, theory of operations, ball assignments, packaging, and electrical specifications. This chapter includes a DSTni block diagram.
- *Section 2: Microprocessor*
  Describes the DSTni microprocessor and its control registers.
- *Section 2: SDRAM*
  Describes the DSTni SDRAM and the registers associated with it.
- *Section 3: Serial Ports*
  Describes the DSTni serial ports and the registers associated with them.
- *Section 3: Programmable Input/Output*
  Describes DSTni's Programmable Input/ Output (PIO) functions and the registers associated with them.
- *Section 3: Timers*
  Describes the DSTni timers.
- *Section 4: Ethernet Controllers*
  Describes the DSTni Ethernet controllers.
- *Section 4: Ethernet PHY*
  Describes the DSTni Ethernet physical layer core.
- *Section 5: SPI Controller*
  Describes the DSTni Serial Peripheral Interface (SPI) controller.
- *Section 5: I2C Controller*
  Describes the DSTni I^{2}C controller.
- *Section 5: USB Controller*
  Describes the DSTni USB controller.
- *Section 5: CAN Controllers*
  Describes the DSTni Controller Area Network (CAN) bus controllers.
- *Section 6: Interrupt Controller*
  Describes the DSTni interrupt controller.
- *Section 6: Miscellaneous Registers*
  Describes DSTni registers not covered in other chapters of this Guide.
- *Section 6: Debugging In-circuit Emulator (Delce)*
- *Section 6: Packaging and Electrical*
  Describes DSTni's packaging and electrical characteristics.
- *Section 6: Applications*
  Describes DSTni's packaging and electrical characteristics.
- *Section 6: Instruction Clocks*
  Describes the DSTni instruction clocks.
- *Section 6: DSTni Sample Code*
- *Section 6: Baud Rate Calculations*
  Provides baud rate calculation tables.

2: Interrupt Controller

This chapter describes the DSTni interrupt controller. Topics in this chapter include:

Overview

DSTni can receive interrupt requests from a variety of internal and external sources. DSTni's internal interrupt controller arranges these requests by priority and presents them one at a time to the microprocessor.

There are 15 interrupt sources available on DSTni:

- The timers use three.
- The UARTs use four.
- The DMA channels use four.
- The peripherals use four:
  - **INT0** connects to Ethernet MAC 0.
  - **INT1** is Ethernet MAC 1 ORed with external 1.
  - **INT2** connects to both the SPI controller and the I^2C controller.
  - **INT3** connects to both the USB controller and an external input pin.
  - **INT6** is both CAN channels.

Interrupts are automatically disabled when an interrupt is taken. Interrupt-service routines (ISRs) can re-enable interrupts by setting the IF flag. This allows interrupts of equal or greater priority to interrupt the currently executing ISR. Interrupts from the same source are disabled so long as the corresponding bit in the interrupt in-service register is set.

Theory of Operation

Interrupt Vector Table

Table 2-1 provides information about the reserved interrupts.

Table 2-1. Interrupt Vectors

Interrupt Name	Vector Type	Vector Address	Default Priority	Related Instructions
Divide Error Exception (See Note 1)	0	00h	1	DIV, DIV
Single Step Interrupt (See Note 2)	1	04h	1A	All
Non-Maskable (NMI)	2	08h	1	INT
Breakpoint Interrupt (See Note 1)	3	0Ch	1	INT
INT0 Detected Overflow Exception	4	10h	1	INT0
Array Bounds Exception (See Note 1)	5	14h	1	BOUND
Unused Opcode Exception (See Note 1)	6	18h	1	Undefined Opcodes
ESC Opcode Exception (See Note 1)	7	1Ch	1	ESC
Time 0 Interrupt (See Note 3)	8	20h	2A	
Reserved	9	24h		
DMA 0 Interrupt	10	28h	4	
DMA 1 Interrupt	11	2Ch	5	
Ethernet MAC 0 (INT0) Interrupt	12	30h	6	
INT1 or Ethernet MAC 1 Interrupt	13	34h	7	
INT2 or SPI/I^{2}C Interrupt	14	38h	8	
INT3 or USB Interrupt	15	3Ch	9	
UART 2 Interrupt	16	40h	10	
UART 1 Interrupt	17	44h	15	
Timer 1 Interrupt (See Note 3)	18	48h	2B	
Timer 2 Interrupt (See Note 3)	19	4Ch	2C	
UART 0 Interrupt	20	50h	15	
INT5 or UART 3 Interrupt	21	54h	11	
DMA 2 Interrupt	22	58h	12	
DMA 3 Interrupt	23	5Ch	13	
CAN Interrupts	24	60h	14	

Default priorities for interrupt sources are used only if you do not program each source to a unique priority level.
Note 1. Generated as a result of an instruction execution.
Note 2. Performed the same way as the 8086.
Note 3. All three timers make up a single interrupt request from the interrupt controller and share the same priority level. However, each timer has a defined priority with respect to the other:
Priority level 2A is the highest, followed by 2B and 2C.

Interrupt Type

An 8-bit interrupt type identifies each of the 256 possible interrupts.

Software exceptions, internal peripherals, and non-cascaded external interrupts supply the interrupt type through the internal interrupt controller.

Cascaded external interrupts and slave-mode external interrupts get the interrupt type from the external interrupt controller by means of interrupt acknowledge cycles on the bus.

Interrupt Vector Table

The interrupt vector table is a 1K memory area that starts at address 00000h. It has up to 256 four-byte address pointers containing the address for the interrupt service routine for each possible interrupt type. For each interrupt, an 8-bit interrupt type identifies the appropriate interrupt vector table entry.

Interrupts 00h to 5Ch are reserved (see Table 2-1 on page 7).

The microprocessor calculates the index to the interrupt vector table by shifting the interrupt type left two bits (multiplying by 4).

Maskable/Nonmaskable Interrupts

Interrupt types 08h through 1Fh are maskable. Of these, only 08h through 14h are actually used (see Table 2-1 on page 7) The maskable interrupts are enabled and disabled by the interrupt enable flag (IF) in the microprocessor status flags; however, the INT command can execute any interrupt regardless of the setting of IF.

Interrupt types 00h through 07h and all software interrupts (the INT instruction) are nonmaskable. The nonmaskable interrupts are not affected by the setting of the IF flag.

DSTni provide two ways to mask and unmask maskable interrupt sources.

- Each interrupt source has an interrupt control register that contains a mask bit specific to that interrupt.
- In addition, the interrupt mask register is provided as a single source to access all of the mask bits.

If the interrupt mask register is written while interrupts are enabled, an interrupt can occur while the register is in an undefined state. This can cause interrupts to be accepted even though they were masked before and after the write to the interrupt mask register. As a result, the interrupt mask register should only be written when interrupts are disabled. Mask bits in the individual interrupt control registers can be written while interrupts are enabled, without erroneous interrupt operation.

Interrupt Enable Flag

The interrupt enable flag (IF) is part of the microprocessor status flags.

- If IF = 1, maskable interrupts are enabled and can cause microprocessor interrupts. (Individual maskable interrupts can still be disabled by means of the mask bit in each control register.)
- If IF = 0, all maskable interrupts are disabled.

The IF flag does not affect the NMI or software exception interrupts (interrupt types 00h to 07h) or the execution of any interrupt through the INT instruction.

Interrupt Mask Bit

Each interrupt control register for the maskable interrupts contains a mask bit (MSK).

If MSK = 1 for a particular interrupt, that interrupt is disabled, regardless of the IF setting.

Interrupt Priority

The column titled Default Priority in Table 2-1 on page 7 shows the priority for the interrupts at power-on reset. The nonmaskable interrupts 00h through 07h are always prioritized ahead of the maskable interrupts.

To reprioritize the maskable interrupts, reconfigure the PR2–PR0 bits in the interrupt control registers. The PR2–PR0 bits in all the maskable interrupts are set to priority level 7 at power-on reset.

Software Interrupts

Software interrupts can be initiated by the INT instruction. Any of the 256 possible interrupts can be initiated by the INT instruction.

- INT 21h causes an interrupt to the vector located at 00084h in the interrupt vector table.
- INT FFh causes an interrupt to the vector located at 003FCh in the interrupt vector table.

Software interrupts are not maskable and are not affected by the setting of the IF flag.

Software Exceptions

A software exception interrupt occurs when an instruction causes an interrupt due to a condition in the microprocessor. Interrupt types 00h, 01h, 03h, 04h, 05h, 06h, and 07h are software exception interrupts.

Software exceptions are not maskable and are not affected by the setting of the IF flag.

Interrupt Conditions and Sequence

The following sections describe how interrupts are serviced.

Nonmaskable Interrupts

The following nonmaskable interrupts are serviced, regardless of the setting of the interrupt enable flag (IF) in the microprocessor status flags.

- The trace interrupt
- The NMI interrupt
- Software interrupts, both user-defined (INT) and software exceptions.

Maskable Hardware Interrupts

- For maskable hardware interrupt requests to be serviced:
- The STI instruction must set the IF flag must be set, and
- The mask bit associated with each interrupt must be reset

Interrupt Request

When an interrupt is requested, DSTni's internal interrupt controller verifies that the interrupt is enabled and that there are no higher priority interrupt requests being serviced or pending.

If the interrupt request is granted, the interrupt controller uses the interrupt type to access a vector from the interrupt vector table (see Table 2-1 on page 7).

Each interrupt type has a four-byte vector available in the interrupt vector table. The interrupt vector table is located in the 1024 bytes from 00000h to 003FFh. Each four-byte vector consists of a 16-bit offset (IP) value and a 16-bit segment (CS) value. The 8-bit interrupt type is shifted left 2 bit positions (multiplied by 4) to generate the index into the interrupt vector table.

Interrupt Servicing

A valid interrupt transfers execution to a new program location based on the vector in the interrupt vector table. The next instruction address (CS:IP) and the microprocessor status flags are pushed onto the stack.

The interrupt enable flag (IF) clears after the microprocessor status flags are pushed on the stack, disabling maskable interrupts during the interrupt service routine (ISR).

The segment:offset values from the interrupt vector table are loaded into the code segment (CS) and the instruction pointer (IP), and execution of the ISR begins.

Returning from an Interrupt

The interrupt return (IRET) instruction pushes the microprocessor status flags and the return address off the stack. Program execution resumes at the point where the interrupt occurred.

The interrupt enable flag (IF) is restored by the IRET instruction along with the remaining microprocessor status flags. If the IF flag was set before the interrupt was serviced, interrupts are re-enabled when the IRET is executed. If there are valid interrupts pending when the IRET is executed, the instruction at the return address is not executed. Instead, the new interrupt is serviced immediately.

If an ISR intends to modify the value of any of the saved flags permanently, it must modify the copy of the microprocessor status flags register that was pushed onto the stack.

Interrupt Priority

Table 2-1 on page 7 shows DSTni's predefined interrupt types and default priority structure. Nonmaskable interrupts (interrupt types 0–7) always have a higher priority than maskable interrupts. However, maskable interrupts have a programmable priority that can override the default priorities relative to one another.

The levels of interrupt priority are:

- Interrupt priority for nonmaskable interrupts and software interrupts
- Interrupt priority for maskable hardware interrupts

Nonmaskable Interrupts and Software Interrupt Priority

The nonmaskable interrupts from 00h to 07h and software interrupts (INT instruction) always take priority over the maskable hardware interrupts. Within the nonmaskable and software interrupts, the trace interrupt has the highest priority, followed by the NMI interrupt, and the remaining nonmaskable and software interrupts.

After the trace interrupt and the NMI interrupt, the remaining software exceptions are mutually exclusive and can only occur one at a time, obviating the need for a further priority breakdown.

Maskable Hardware Interrupt Priority

Starting with interrupt type 8 (timer 0 interrupt), the maskable hardware interrupts have both a default priority (see Table 2-1 on page 7) and a programmable priority. The programmable priority is the primary priority for maskable hardware interrupts. The overall priority is the secondary priority for maskable hardware interrupts.

Since all maskable interrupts are set to a programmable priority of seven on reset, the overall priority of the interrupts determines the priority in which each interrupt is granted by the interrupt controller until programmable priorities are changed by reconfiguring the control registers.

The default priority levels shown in Table 2-1 on page 7 are not the same as the programmable priority level associated with each maskable hardware interrupt. Each of the maskable hardware interrupts has a programmable priority from 0 to 7, with 0 being the highest priority (see Table 2-1 on page 7).

For example, if the INT4–INT0 interrupts are all changed to programmable priority 6 and no other programmable priorities are changed from the reset value of seven, the INT4–INT0 interrupts take precedence over all other maskable interrupts. (Within INT4–INT0, the hierarchy is as follows: INT0>INT1>INT2>INT3>INT4.)

Software Exceptions, Traps, and NMI

The following predefined interrupts cannot be masked by programming.

Divide Error Exception (Interrupt Type 00h)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of destination bits.

Trace Interrupt (Interrupt Type 01h)

If the trace flag (TF) in the microprocessor status flags register is set, the trace interrupt is generated after most instructions. This interrupt lets program execute in single-step mode. The interrupt is not generated after prefix instructions like REP, instructions that modify segment registers like POP DS, or the WAIT instruction.

Taking the trace interrupt clears the TF bit after the microprocessor status flags are pushed onto the stack. The IRET instruction at the end of the single step interrupt service routine restores the microprocessor status flags (and the TF bit) and transfers control to the next instruction to be traced.

Trace mode is initiated by pushing the microprocessor status flags onto the stack, then setting the TF flag on the stack, and then popping the flags.

Nonmaskable Interrupt-NMI (Interrupt Type 02h)

This pin tells DSTni that an interrupt request has occurred. The NMI signal is the highest priority hardware interrupt and, unlike the INT4–INT0 pins, cannot be

masked. DSTni always transfers program execution to the location specified by the nonmaskable interrupt vector in the DSTni interrupt vector table when NMI is asserted.

Although NMI is the highest priority interrupt source, it does not participate in the priority resolution process of the maskable interrupts. There is no bit associated with NMI in the interrupt in-service or interrupt request registers. This means that a new NMI request can interrupt an executing NMI interrupt service routine. As with all hardware interrupts, the IF (interrupt flag) clears when the microprocessor takes the interrupt, disabling the maskable interrupt sources. However, if maskable interrupts are re-enabled by software in the NMI interrupt service routine (via the STI instruction, for example), the NMI currently in service does not affect the priority resolution of maskable interrupt requests. For this reason, the NMI interrupt service routine should not enable the maskable interrupts.

Breakpoint Interrupt (Interrupt Type 03h)

An interrupt caused by the 1-byte version of the INT instruction (INT3).

INT0 Detected Overflow Exception (Interrupt Type 04h)

Generated by an INT0 instruction if the OF bit is set in the Processor Status Flags (F) register.

Array BOUNDS Exception (Interrupt Type 05h)

Generated by a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands.

The other operand indicates the value of the index to be checked.

Unused Opcode Exception (Interrupt Type 06h)

Generated if execution is attempted on undefined opcodes.

ESC Opcode Exception (Interrupt Type 07h)

Generated if execution of ESC opcodes (D8h–DFh) is attempted. DSTni does not check the escape opcode trap bit. The return address of this exception points to the ESC instruction that caused the exception. If a segment override prefix preceded the ESC instruction, the return address points to the segment override prefix.

Note: *All numeric coprocessor opcodes cause a trap. DSTni does not support the numeric coprocessor interface.*

Interrupt Acknowledge

Interrupts can be acknowledged in two different ways:

- The internal interrupt controller can provide the interrupt type.
- An external interrupt controller can provide the interrupt type.

The microprocessor requires the interrupt type as an index into the interrupt vector table. When the internal interrupt controller is supplying the interrupt type, no interrupt acknowledge bus cycles are generated. The only external indication that an interrupt is being serviced is the microprocessor reading the interrupt vector table.

When an external interrupt controller supplies the interrupt type, the microprocessor generates two interrupt acknowledge bus cycles. The external interrupt controller writes the interrupt type to the AD7–AD0 lines during the second bus cycle.

Interrupt acknowledge bus cycles have the following characteristics:

- The two interrupt acknowledge cycles are locked.
- Two idle states are always inserted between the two interrupt acknowledge cycles.
- Wait states are inserted if READY is not returned to the microprocessor.

Interrupt Controller Reset Conditions

On reset, the interrupt controller performs the following actions:

1. All special fully nested mode (SFNM) bits are reset, implying fully nested mode.
2. All priority (PR) bits in the various control registers are set to 1. This places all sources at the lowest priority (level 7).
3. All level-triggered mode (LTM) bits are reset to 0, resulting in edge-triggered mode.
4. All interrupt in-service bits are reset to 0.
5. All interrupt request bits are reset to 0.
6. All mask (MSK) bits are set to 1. All interrupts are masked.
7. All cascade (C) bits are reset to 0 (non-cascade).
8. The interrupt priority mask is set to 7, permitting interrupts of all priorities.
9. The interrupt controller is initialized to master mode.

Polled Environments

The interrupt controller can be used in polled mode if interrupts are not desired. When polling, interrupts are disabled and software polls the interrupt controller as required. The interrupt controller is polled by reading the Poll Status register (see Poll Status Register on page 29).

- Bit [15] indicates to the microprocessor that an interrupt of high enough priority is requesting service.
- Bits [4:0] indicate to the microprocessor the interrupt type of the highest-priority source requesting service.

After determining that an interrupt is pending, software reads the Poll register (see Poll Register on page 29), which causes the in-service bit of the highest-priority source to be set.

To enable reading of the Poll register information without setting the indicated in-service bit, DSTni provides a Poll Status register in addition to the Poll register. The Poll Status register contains the same information in the Poll register; however the Poll Status register can be read without setting the associated in-service bit. These registers are located in two adjacent memory locations in the peripheral control block.

End-of-Interrupt Write to the EOI Register

When an interrupt service routine completes, a program must write to the EOI register to reset the in-service (IS) bit. There are two types of writes to the EOI register — specific EOI and non-specific EOI (see End-of-Interrupt Write to the EOI Register on page 14).

Non-specific EOI does not specify which IS bit is to be reset. Instead, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine.

Specific EOI requires the program to send the interrupt type to the interrupt controller to indicate the source IS bit that is to be reset. Specific reset is applicable when interrupt nesting is possible or when the highest priority IS bit that was set does not belong to the service routine in progress.

Interrupt Controller Register Summary

Table 2-2. Interrupt Controller Register Summary

Hex Offset	Description	Page
4C	CAN Interrupt Control register	15
4A	DMA 3 Interrupt Control register	15
48	DMA 2 Interrupt Control register	16
46	Serial Port 3 Interrupt Channel register	16
44	Serial Port 0 Interrupt Channel register	17
42	Serial Port 1 Interrupt Channel register	17
40	Serial Port 2 Interrupt Channel register	18
3E	INT3 or USB Interrupt Control register	19
3C	INT2 (SPI/I^2C) Interrupt Control register	19
3A	INT1 or Ethernet MAC 1 Interrupt Control register	21
38	Ethernet MAC 0 (INT0) Interrupt Control register	22
36	DMA 1 Interrupt Control register	23
34	DMA 0 Interrupt Control register	23
32	Timer Interrupt Control register	23
30	Interrupt Status register	24
2E	Interrupt Request register	25
2C	In-Service register	26
2A	Priority Mask register	27
28	Interrupt Mask register	28
26	Poll Status register	29
24	Poll register	29
22	End of Interrupt (EOI) register	30

Register Definitions

CAN Interrupt Control Register

Table 2-3. CAN Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	4Ch															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-4. CAN Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

DMA 3 Interrupt Control Register

Table 2-5. DMA 3 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	4Ah															
FIELD	///												MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-6. DMA 3 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

DMA 2 Interrupt Control Register

Table 2-7. DMA 2 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	48h															
FIELD													MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-8. DMA 2 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Serial Port 3 Interrupt Control Register

Table 2-9. Serial Port 3 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	46h															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-10. Serial Port 3 Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Serial Port 0 Interrupt Control Register

Table 2-11. Serial Port 0 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	44h															
FIELD	///												MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-12. Serial Port 0 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Serial Port 1 Interrupt Control Register

Table 2-13. Serial Port 1 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	42h															
FIELD	///												MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-14. Serial Port 1 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Serial Port 2 Interrupt Control Register

Table 2-15. Serial Port 2 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	40h															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-16. Serial Port 2 Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

INT3 or USB Interrupt Control Register

Table 2-17. INT3 or USB Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	3Eh															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-18. Serial Port 2 Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

INT2 (SPI/I^{2}C) Interrupt Control Register

Table 2-19. INT2 (SPI/I^{2}C) Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	3Ch															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-20. INT2 (SPI/I^{2}C) Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

INT1 or Ethernet MAC 1 Interrupt Control Register

Table 2-21. INT1 or Ethernet MAC 1 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	3Ah															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-22. INT1 or Ethernet MAC 1 Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Ethernet MAC 0 (INT0) Interrupt Control Register

Table 2-23. Ethernet MAC 0 (INT0) Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	38h															
FIELD	///											LTM	MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-24. Ethernet MAC 0 (INT0) Interrupt Control Register Definitions

Bits	Field Name	Description
15:5	\\\	**Reserved**
4	LTM	**Level Trigger Mode** Sets the respective interrupt source. 1 = enable level-triggered mode. An interrupt generates when the external interrupt signal is HIGH. 0 = enable edge-triggered mode (*default*). For both settings, the level must remain HIGH until the interrupt is acknowledged.
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

DMA 1 Interrupt Control Register

Table 2-25. DMA 1 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	36h															
FIELD													MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-26. DMA 1 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

DMA 0 Interrupt Control Register

Table 2-27. DMA 0 Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	34h															
FIELD													MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	R W	RW	R W	RW	RW	RW	RW	R W	R W	R W	R W	RW	R W	R W	R W	R W

Table 2-28. DMA 0 Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Timer Interrupt Control Register

Table 2-29. Timer Interrupt Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	32h															
FIELD	///												MSK	PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-30. Timer Interrupt Control Register Definitions

Bits	Field Name	Description
15:4	\\\	**Reserved**
3	MSK	**Mask Interrupt** 1 = mask respective interrupt request (*default*) 0 = enable respective interrupts.
2:0	PR[2:0]	**Programmable Priority Level** The programmable priority level for the respective interrupt source. 111 = lowest priority.(*default*) 000 = highest priority.

Interrupt Status Register

Table 2-31. Interrupt Status Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	30h															
FIELD	DHLT	///												IRT2	IRT1	IRT0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-32. Interrupt Status Register Definitions

Bits	Field Name	Description
15	DHLT	**Halt DMA Operations** 1 = halt all DMA operations. 0 = do not halt all DMA operations (*default*). Automatically set when an NMI occurs, and resets when an IRET instruction executes. By suspending DMA operations during an NMI, the microprocessor can quickly service the NMI request. Programmers can also set this bit.
14:3	\\\	**Reserved**
2:0	IRT[2:0]	**Timer Interrupt Request Bits** Lets software differentiate between timer interrupts, as the TMR bit in the Interrupt Request register is the logical OR of all timer requests. Setting any of these bits generates a timer-interrupt request.

Interrupt Request Register

Table 2-33. Interrupt Request Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	2Eh															
FIELD	///	I6	D3	D2	SP 3	SP 0	SP 1	SP 2	I3	I2	I1	I0	D1	D0	///	TMR
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	R

Table 2-34. Interrupt Request Register Definitions

Bits	Field Name	Description
15	\\\	**Reserved**
14	I6	**Logical "OR" Connected to Both CAN0 and CAN1** This bit is the in-service bit for this interrupt source.
13:12	D[3:0]	**Interrupt Request Bits for the DMA Channels (DMA3:0)** Setting any of these bits generates an interrupt request on the corresponding DMA channel interrupt request line. Resetting any of these bits removes the interrupt request.
11:8	SP[3:0]	**State of the Asynchronous Serial Port Interrupt Requests** These bits are set when the respective serial port generates an interrupt request. These bits clear when the respective interrupt acknowledge cycle occurs. D3:0 In-service bits for DMA channels DMA3:0. I0 Logical "OR" connected to both internal MACs. This bit is the in-service bit.
7	I3	**Logical "OR" Connected to External Interrupt 3 and the USB Controller** This bit is the in-service bit for this interrupt source.
6	I2	**Logical "OR" Connected to the SPI Controller and the I^{2}C Controller** This bit is the in-service bit for this interrupt source.
5	I1	**Logical "OR" Connected to External Interrupt 1 and Ethernet MAC 1** This bit is the in-service bit for this interrupt source.
4	I0	**Ethernet MAC 0** This bit is the in-service bit for this interrupt source.
3:2	D[1:0]	**In-Service Bits for DMA Channels DMA1:0**
1	///	**Reserved**
0	TMR	**Logical OR of All Timer Interrupt Requests** The individual timer interrupt request bits are contained in the interrupt status register. This bit cannot be written.

In-Service Register

Table 2-35. In-Service Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	2Ch															
FIELD	///	I6	D3	D2	SP 3	SP 0	SP 1	SP 2	I3	I2	I1	I0	D1	D0	///	TMR
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-36. In-Service Register Definitions

Bits	Field Name	Description
15	\\\	**Reserved**
14	I6	**Logical "OR" Connected to Both CAN0 and CAN1** This bit is the in-service bit for this interrupt source.
13:12	D[3:0]	**Interrupt Request Bits for the DMA Channels (DMA3:0)** Setting any of these bits generates an interrupt request on the corresponding DMA channel interrupt request line. Resetting any of these bits removes the interrupt request.
11:8	SP[3:0]	**State of the Asynchronous Serial Port Interrupt Requests** These bits are set when the respective serial port generates an interrupt request. These bits clear when the respective interrupt acknowledge cycle occurs. D3:0 In-service bits for DMA channels DMA3:0. I0 Logical "OR" connected to both internal MACs. This bit is the in-service bit.
7	I3	**Logical "OR" Connected to External Interrupt 3 and the USB Controller** This bit is the in-service bit for this interrupt source.
6	I2	**Logical "OR" Connected to the SPI Controller and the I^{2}C Controller** This bit is the in-service bit for this interrupt source.
5	I1	**Logical "OR" Connected to External Interrupt 1 and Ethernet MAC 1** This bit is the in-service bit for this interrupt source.
4	I0	**Ethernet MAC 0** This bit is the in-service bit for this interrupt source.
3:2	D[1:0]	**In-Service Bits for DMA Channels DMA1:0**
1	///	**Reserved**
0	TMR	**Logical OR of All Timer Interrupt Requests** The individual timer interrupt request bits are contained in the interrupt status register. This bit cannot be written.

Priority Mask Register

Table 2-37. Priority Mask Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	2Ah															
FIELD	///													PR2	PR1	PR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-38. Priority Mask Register Definitions

Bits	Field Name	Description
15:3	\\\	**Reserved**
2:1	PR[2:1]	**Minimum Priority Level an Interrupt Request Must Have to be Recognized** An interrupt request is processed by the interrupt controller if its priority level is greater than or equal to the priority in this register. 111 = lowest priority (*default*). 000 = highest priority.
0	TMR	**Logical OR of All Timer Interrupt Requests** The individual timer interrupt request bits are contained in the interrupt status register. This bit cannot be written.

Interrupt Mask Register

Table 2-39. Interrupt Mask Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	28h															
FIELD	///	I6	D3	D2	SP 3	SP 0	SP 1	SP 2	I3	I2	I1	I0	D1	D0	///	TMR
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
RW	R W	R W	R W	R W	R W	R W	R W	R W	R W	R W	R W	R W	R W	R W	RW	RW

Table 2-40. Interrupt Mask Register Definitions

Bits	Field Name	Description
15	\\\	**Reserved**
14	I6	**Logical "OR" Connected to Both CAN0 and CAN1** 1 = mask CAN 0 and CAN 1 (*default*). 0 = do not mask CAN 0 and CAN 1.
13	D3	**Mask DMA Channel 3 Interrupt** 1 = mask DMA channel 3 (*default*). 0 = do not mask DMA channel 3.
12	D2	**Mask DMA Channel 2 Interrupt** 1 = mask DMA channel 2 (*default*). 0 = do not mask DMA channel 2.
11	SP3	**Asynchronous Serial Port 3** 1 = mask asynchronous serial port 3 (*default*). 0 = do not mask asynchronous serial port 3.
10	SP0	**Asynchronous Serial Port 0** 1 = mask asynchronous serial port 0 (*default*). 0 = do not mask asynchronous serial port 0.
9	SP1	**Asynchronous Serial Port 1** 1 = mask asynchronous serial port 1 (*default*). 0 = do not mask asynchronous serial port 1.
8	SP2	**Asynchronous Serial Port 2** 1 = mask asynchronous serial port 2 (*default*). 0 = do not mask asynchronous serial port 2.
7	I3	**Logical "OR" Connected to External Interrupt 3 and USB Controller** 1 = mask external interrupt 3 and USB controller (*default*). 0 = do not mask external interrupt 3 and USB controller.
6	I2	**Logical "OR" Connected to the SPI Controller and the I^{2}C Controller** 1 = mask the SPI controller and I^{2}C controller (*default*). 0 = do not mask the SPI controller and I^{2}C controller.
5	I1	**Logical "OR" Connected to External Interrupt 1 and Ethernet MAC 1** 1 = mask external interrupt 1 and Ethernet MAC 1 (*default*). 0 = do not mask external interrupt 1 and Ethernet MAC 1.
4	I0	**Ethernet MAC 0** 1 = mask Ethernet MAC 0 (*default*). 0 = do not mask Ethernet MAC 0.
3	D1	**Mask DMA Channel 1 Interrupt** 1 = mask DMA channel 1 (*default*). 0 = do not mask DMA channel 1.
2	D0	**Mask DMA Channel 0 Interrupt** 1 = mask DMA channel 0 (*default*). 0 = do not mask DMA channel 0.
1	///	**Reserved**
0	TMR	**Logical OR of All Timer Interrupt Requests** The individual timer interrupt request bits are contained in the interrupt status register. This bit cannot be written.

Poll Status Register

Note: *These bits are only valid if IRQ=1.*

Differences between the Poll Status and Poll registers:

- Reading the Poll register generates a software poll. This sets the in-service bit for the highest priority-pending interrupt.
- Reading the Poll Status register does not set the in-service bit for the highest priority-pending interrupt.

Table 2-41. Poll Status Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	24h															
FIELD	IRQ	///														
RESET	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-42. Poll Status Register Definitions

Bits	Field Name	Description
15	IRQ	**Pending Interrupt** Determines whether an interrupt request is pending. 1 = interrupt request is present. 0 = interrupt request is reset (*default*).
14:5	///	**Reserved**
4:0	S[4:0]	**Highest Priority Interrupt Source** Contain the encoded vector type of the highest priority interrupt source.

Poll Register

Table 2-43. Poll Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	24h															
FIELD	IRQ	///										S4	S3	S2	S1	S0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	R W	R W	R W	R W	RW	RW	RW	RW	R W	R W	R W	R W	R W	R W	R W	RW

Table 2-44. Poll Register Definitions

Bits	Field Name	Description
15	IRQ	**Pending Interrupt** Determines whether an interrupt request is pending. 1 = interrupt request is present. 0 = interrupt request is reset (*default*).
14:5	///	**Reserved**
4:0	S[4:0]	**Highest Priority Interrupt Source** Contain the encoded vector type of the highest priority interrupt source.

End of Interrupt Register

Table 2-45. End of Interrupt Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	22h															
FIELD	SPC	///										S4	S3	S2	S1	S0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Table 2-46. End of Interrupt Register Definitions

Bits	Field Name	Description
15	SPC	**Type of EOI command** 1 = issue a non-specific EOI command in S4:S0. 0 = issue a specific EOI command in S4:S0 (*default*).
14:5	///	**Reserved**
4:0	S[4:0]	**Highest Priority Interrupt Source** Contain the encoded vector type of the highest priority interrupt source.

3: Miscellaneous Registers

Table 3-1. Miscellaneous Register Summary

Hex Offset	Mnemonic	Register Description	Page
AE	CAR	Checksum Adder register	32
AC	CDR	Checksum Data register	32
7E	LEDC	LED Control register	33
6E	PLLCLK	Phase Lock Loop, Clock register	35
6A	RNG	Random Number Generator register	38

Checksum Adder Register

Always write a value to the Checksum Data register to initialize it before using the Checksum Adder Register.

Note: *The Checksum Adder register is a single hardware resource that must be protected from being accessed simultaneously by multiple application threads.*

Table 3-2. Checksum Adder Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	AEh															
FIELD	DATA [7:0]								DATA [15:8]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	R W	RW	R W	R W	RW	RW	RW	RW	R W	R W	R W	R W	R W	R W	R W	R W

Table 3-3. Checksum Adder Register Definitions

Bits	Field Name	Description
15:0	DATA [15:0]	Writing to this register adds the byte swapped data to the Checksum register with carry. The data is byte-swapped during this write. This register is to be used with TCPIP checksum generation. Reading this register shows the data in the adder.

Checksum Data Register

Table 3-4. Checksum Data Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	ACh															
FIELD	DATA [15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	R W	RW	RW	RW	RW	RW	RW	R W	R W	R W	R W	R W	R W	R W	R W	R W

Table 3-5. Checksum Data Register Definitions

Bits	Field Name	Description
15:0	DATA [15:0]	Writing to this register sets the checksum value. Typically this can be a starting value. Reading this register shows its current value after any writes to the Checksum Adder register.

LED Control Register

The LEDs normally connect to four control outputs from the internal PHY. To control the LEDs by software, set SEN to 1.

Note: *The reset value for this register, 0000h, is read as 000Dh because the LED signals initially are driven from the PHY.*

Table 3-6. LED Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	7Eh															
FIELD	///										ENC	SEN	LED3	LED2	LED1	LED0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	R	R	R	R	R	R	R	R	R	R	RW	RW	RW	RW	RW	RW

Table 3-7. LED Control Register Definitions

Bits	Field Name	Description
15:6	///	**Reserved** Read only as 0.
5	ENC	**Encoding** 0 = the LED signals are driven directly with no encoding. 1 = the LED signals are driven through an encoder to allow connection to two wire Bi-color LED's. The un-encoded signals are active LOW. The encoded signals are shown with an E in front.
4	SEN	0 = the LED signals are driven from the PHY. 1 = the LED signals are driven from this register.
3	LED3	**LED3 Control Line** Normally, this bit connects to duplex signal. If SEN = 0, the LED3 signal is driven from the PHY and reading LED3 indicates the PHY status. When duplex is 0, the PHY is in full-duplex mode. If SEN = 1, the LED3 signal is driven from this register. See Table 3-8.
2	LED2	**LED2 Control Line** Normally, this bit connects to the activity signal. If SEN = 0, the LED2 signal is driven from the PHY and reading LED2 indicates the PHY status. Activity is 0 when the PHY detects or generates valid Ethernet traffic. If SEN = 1, the LED2 signal is driven from this register. See Table 3-8.
1	LED1	**LED1 Control Line** Normally, this bit connects to the link signal. If SEN = 0, the LED1 signal is driven from the PHY and reading LED1 indicates the PHY status. Link is 0 when the PHY has a valid link. If SEN = 1, the LED1 signal is driven from this register. See Table 3-9
0	LED0	**LED0 Control Line** Normally, this bit connects to the 100Mbit signal. If SEN = 0, the LED0 signal is driven from the PHY and reading LED0 indicates the PHY status. 100Mbit is 0 when in 100Mbit mode. If SEN = 1, the LED0 signal is driven from this register. See Table 3-9.

Table 3-8. LED Bits [3] and [2]

LED3 Duplex	LED2 Activity	ELED2 Green (R)	ELED3 Yellow (R)	Function
1	1	0	0	No activity
1	0	0	1	Half-duplex
0	1	0	0	No activity
0	0	1	0	Full-duplex

Table 3-9. LED Bits [1] and [0]

LED1 Link	LED0 100Mbps	ELED0 Green (L)	ELED1 Yellow (L)	Function
1	1	0	0	No link
1	0	0	0	No link
0	1	0	1	10 Mbit link
0	0	1	0	100 Mbit link

Note: *The ELED signals in Table 3-8 and Table 3-9 stand for Encoded LEDs and are enabled by ENC bit [5] in the LED Control register. The other LED signals in these tables apply when ENC is off.*

PLL/CLK Control Register

PLL/CLK Control is the Phase Lock Loop/Clock Control register.

Table 3-10. PLL/CLK Control Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	6Eh															
FIELD	///	///	PLL BYP N	LOCKED	PLLMULT								USBDIV			
RESET	0	0	—	—	0	0	0	1	1	0	0	0	0	0	0	0
RW	RW	RW	R	R	R W	RW	RW	R W	RW	R W	RW	R W	R W	R W	R W	R W
Default	00xx				18h								0			

Table 3-11. PLL/CLK Control Register Definitions

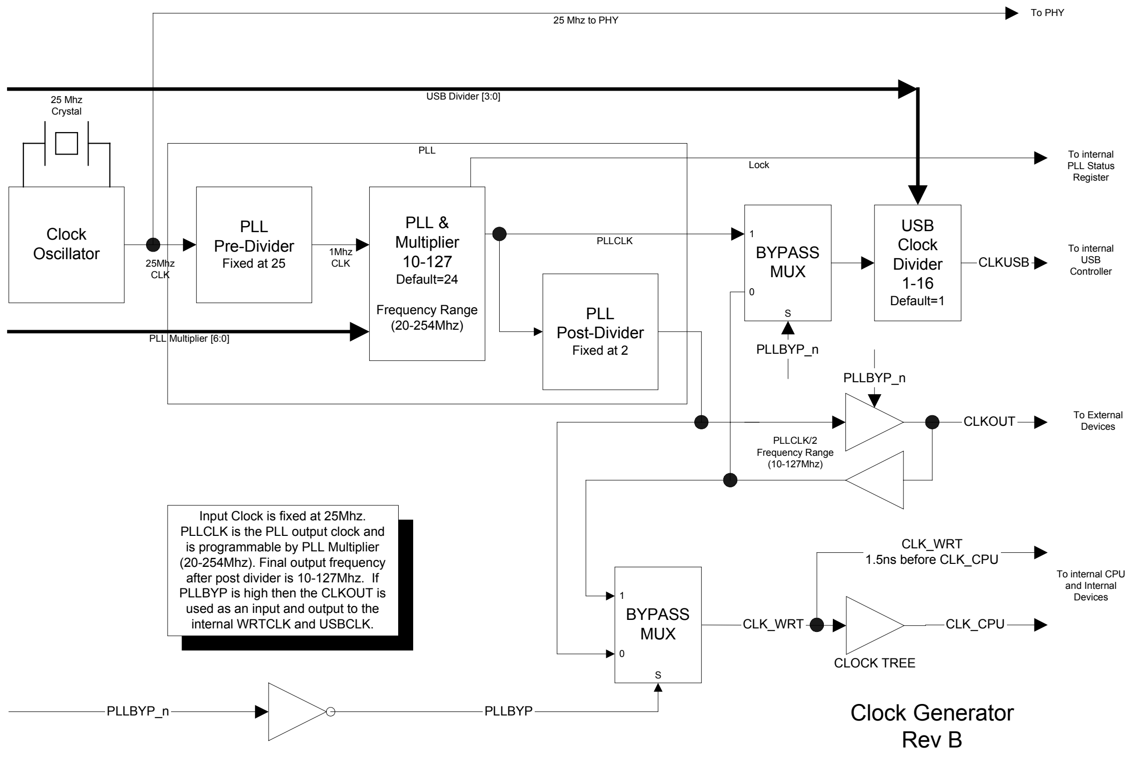
Bits	Field Name	Description
15:14	///	**Reserved**
13	PLLBYP N	**PLL Bypass Pin** 1 = PLL is being used to generate CPUCLK. 0 = PLL is being bypassed and CPUCLK is receiving a clock to use as CPUCLK.
12	LOCKED	**PLL LOCKED** 1 = indicates the PLL has locked onto the desired frequency set by PLLMULT. 0 = PLL is trying to change to the desired frequency. For frequencies below 13 MHz, lock may not be possible because of PLL jitter.
11:4	PLLMULT	**PLL Multiplier** x1= 01h to x7Fh. Default=x18h (24 MHz) The PLL Multiplier sets the value that the PLL uses to multiply the input clock. With a 25 MHz crystal, the frequency will be a multiple of 1 MHz. The maximum clock rate is limited by the CPU cycle time. Exceeding this value causes unpredictable results. The PLL output is connected directly to the CPU Clock, unless the PLLBYP (PLL Bypass is pulled LOW. The clock is sourced from the CLKOUT pin, which is then tri-stated by PLLBYP_n being LOW; in this case, the PLL is not used and should be run as slowly as possible to minimize power. Be aware of the impact the clock frequency has on Flash access time and serial baud rates, and adjust the other appropriate register values.
3:0	USBDIV	**USB Clock Divider** These four bits set the USB clock divider from the PLL output clock frequency to the USB block. The input clock to the USB divider is 2 times the CPUCLK rate when the PLL is used (PLLBYP_n =1). The input to the USB divider is connected to CLKOUT signal directly when PLL is disabled (PLLBYP_n = 0). See Table 3-12. Clock Divider. /1=0000 to /16=1111. Default = /1 ***Note:*** *The USB clock must be 48 MHz for USB to work properly in high-speed mode and 6 MHz for some low-speed modes. This may limit the CPU clock speeds that can be used if the USB is used. If the USB is not used, set the divider to the maximum to minimize power.*

Table 3-12. Divider Bits and Corresponding Values

PLLMULT							VCO Divider	FB Divider	USB Divider	Clock (MHz)	
6	5	4	3	2	1	0	1, 2 or 4	(N)		USB	CPU
0	0	0	0	0	0	0	4	0	-	?	?
0	0	0	0	0	0	1	4	1	-	2	1
0	0	0	0	0	1	0	4	2	-	4	2
0	0	0	0	0	1	1	4	3	-	6	3
0	0	0	0	1	0	0	4	4	-	8	4
0	0	0	0	1	0	1	4	5	-	10	5
0	0	0	0	1	1	0	4	6	-	12	6
0	0	0	0	1	1	1	4	7	-	14	7
0	0	0	1	0	0	0	4	8	-	16	8
0	0	0	1	0	0	1	4	9	-	18	9
0	0	0	1	0	1	0	4	10	-	20	10
0	0	0	1	0	1	1	4	11	-	22	11
0	0	0	1	1	0	0	4	12	-	24	12
:	:	:	:	:	:	:	:	:	-	:	:
0	**0**	**1**	**1**	**0**	**0**	**0**	**4 (Default)**	**24(Default)**	**1**	**48**	**24**
:	:	:	:	:	:	:	:	:	-	:	:
0	0	1	1	1	1	1	4	31	-	62	31
0	1	0	0	0	0	0	2	32	-	64	32
:	:	:	:	:	:	:	:	:	-	:	:
0	1	1	0	0	0	0	2	48	2	96	48
:	:	:	:	:	:	:	:	:	-	:	:
0	1	1	1	1	1	1	2	63	-	126	63
1	0	0	0	0	0	0	1	64	-	128	64
:	:	:	:	:	:	:	:	:	-	:	:
1	0	0	1	0	0	0	1	72	3	144	72
:	:	:	:	:	:	:	:	:	-	:	:
1	0	0	1	0	0	0	1	96	4	192	96
:	:	:	:	:	:	:	:	:	-	:	:
1	1	1	1	0	0	0	1	120	5	240	120
:	:	:	:	:	:	:	:	:	-	:	:
1	1	1	1	0	1	1	1	123	-	246	123
1	1	1	1	1	0	0	1	124	-	248	124
1	1	1	1	1	0	1	1	125	-	250	125
1	1	1	1	1	1	0	1	126	-	252	126
1	1	1	1	1	1	1	1	127	-	254	127

Notes: Gray rows show values that can be used to run the USB at standard 1 2Mbit data rates. The internal 256K bytes of memory use different internal refresh timing. Therefore, when the clock speed exceeds 63 MHz, fewer internal refresh cycles can be performed. This reduces the amount of power used by the internal memory. These refresh cycles are transparent to the user and do not effect memory access speeds. This register controls that timing. If the internal PLL is bypassed and the CPU clock frequency exceeds 63 MHz, this register must be programmed for 64 MHz. If the internal PLL is bypassed and the CPU clock frequency is 63 MHz or less, program this register to 24 MHz. Clock frequencies above 115 MHz are not guaranteed across all DSTni design specifications.

Figure 3-1. PLL and Clock Generator


25 Mhz to PHY
To PHY
USB Divider [3:0]
25 Mhz Crystal
PLL
Lock
To internal PLL Status Register
Clock Oscillator
25Mhz CLK
PLL Pre-Divider Fixed at 25
1Mhz CLK
PLL & Multiplier 10-127 Default=24
Frequency Range (20-254Mhz)
PLLCLK
1
BYPASS MUX
0
S
USB Clock Divider 1-16 Default=1
CLKUSB
To internal USB Controller
PLL Multiplier [6:0]
PLL Post-Divider Fixed at 2
PLLBYP_n
PLLBYP_n
CLKOUT
To External Devices
PLLCLK/2 Frequency Range (10-127Mhz)
CLK_WRT 1.5ns before CLK_CPU
To internal CPU and Internal Devices
1
BYPASS MUX
0
S
CLK_WRT
CLK_CPU
CLOCK TREE
PLLBYP_n
PLLBYP
Clock Generator Rev B

Input Clock is fixed at 25Mhz. PLLCLK is the PLL output clock and is programmable by PLL Multiplier (20-254Mhz). Final output frequency after post divider is 10-127Mhz. If PLLBYP is high then the CLKOUT is used as an input and output to the internal WRTCLK and USBCLK.

Random Number Generator Register

The Random Number Generator register provides a random number for use in the TCPIP or MAC as an address. The random-number generator is a counter running at the current CPU clock frequency and continuously updates on each clock. The data read is a 16-bit data field. Writes to this register are to bits [1] and [0].

Writes to this register control which type number is returned. The random numbers are not affected by these writes. The linear version of the random number can also be used to indicate the number of clocks that have passed for timing some code execution. The linear number consists of two 16-bit registers, with RS=1 being the most-significant 16 bits.

Table 3-13. Random Number Generator Register

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET	6Ah															
FIELD	Don't Care														RS	LI
	DATA[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RW	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R W	R W

Table 3-14. Random Number Generator Register Definitions

Bits	Field Name	Description
15:2	///	**Don't Care**
1	RS	**Register Select** 0 = register 0. 1 = register 1. Only used in Linear Number mode. See Table 3-15.
0	LI	**Linear** 0 = selects Pseudo Random number in a 16-bit register. 1 = selects Linear numbers in a 32-register. See Table 3-15.
15:0	DATA[15:0]	**Random Number Generator Data** A counter running at the current CPU clock frequency that continuously updates on each clock.

Table 3-15. RS/LI Combinations

RS	LI	Description
0	0	Pseudo Random Number (16 bits)
0	1	Linear Clock Counter (lower 16 bits)
1	0	Reserved
1	1	Linear Clock Counter (Upper 16 bits)

4: Debugging In-circuit Emulator (DeIce)

This chapter describes the Debugging In-circuit Emulator (DeIce). Topics in this chapter include:

- Theory of Operation on page 39
- FS2 Target Connection on page 45

Theory of Operation

The CPU has an integrated Joint Test Action Group (JTAG) DeBugger/In-Circuit-Emulator called CPUDICE. CPUDICE integrates an IEEE 1149.1 JTAG Test Access Port (TAP) or “slave“ controller that is typically connected to a JTAG debugger. The CPUDICE architecture has two submodules:

- BRKPTS, which contains the logic for four hardware breakpoints.
- TRCEBUFF, which contains a trace buffer of 256 instructions.

Overview

Generally, users of the CPU will rely on First Silicon Solutions JTAG debugger or Paradigm software to control the CPUDICE core. This section is provided for advanced users who want to understand and extend the CPUDICE architecture features.

The CPUDICE architecture provides the logic to stop, single step, read/write memory, interrogate the state of the CPU. The CPUDICE controls the CPU via JTAG instructions and the scan chains described below.

The CPUDICE controls the CPU at the bus cycle level. This means it can force feed any byte or word into the CPU and stop the processor between any bus cycle. The CPUDICE cannot stop in the middle of instructions, only bus cycles (any memory or IO operation). The CPUDICE can also insert any memory or I/O read or write into the stream while the processor is stopped or while it is running.

The CPUDICE core provides access and control of the CPU via JTAG instructions and scan chains.

ADDR Scan Chain

The ADDR scan chain consists of only the 24 bits of the CPUs address bus. The address bus is setup as a separate chain to allow rapid polling of the address bus during real time operation for generating histograms. The chain is 24 bits long with bit 0 shifted out first and bit 23 shifted out last. The address is the linear address generated after the offset is added to the segment register.

DATA Scan Chain

The DATA scan chain consists of the 16-bit DATA bus and 8 bits of control/status information. These 24 bits are appended to the front of the ADDR scan chain. Bits [2:0] indicate the type of cycle requested or captured as defined in the CPU STATUS signals. Bit [3] has two different meanings, depending on whether data is being captured or cycles are being initiated.

- When bus cycles are being captured, bit [3] is a one when a DMA cycle has been captured.
- When performing MEMREAD, MEMWRITE, IOREAD, or IOWRITE JTAG instructions, program bit [3] with either a 1 when a 16-bit cycle is desired or a zero when an 8-bit cycle is desired.

When using the APPLY_CPU instruction, bits [5] and [4] in the control chain have special meaning. When bit [4] is set and the APPLY_CPU instruction is used, the CPU's instruction queue is flushed before the scanned in vector is applied. Bit [5] is a select for opcodes versus data. When applying data for an opcode fetch, bit [5] should be clear. When applying data for a memory or I/O read (anything but an opcode fetch), bit [5] should be set.

Table 4-1. ADDR, DATA, and STAT Scan Chain

Bits	Field Name	Description
23:0	ADDR[23:0]	**ADDR[23:0]**
39:24	DATA[15:0]	**Data[15:0]**
47:40	STAT[2:0]	**STAT[2:0]** Valid settings are: 000 = interrupt ACK 001 = read IO 010 = write IO 011 = halt 100 = instruction fetch 101= read memory 110 = write memory 111 = idle CTL[3] = sixteen 0 = 8-bit cycle 1 = 16-bit cycle CTL[4] = IQ flush (for APPLYs) CTL[5] = DATA (for APPLYs) The following bits are used only in HW_BKPT chains: CTL[4] = TR_QUAL mode CTL[5] = OP_EXEC mode CTL[7:6] = MODE

STATUS Scan Chain

The STATUS scan chain consists of a single 8-bit register with some status information as listed in the table below.

Table 4-2. STATUS Scan Chain

Bits	Field Name	Description
7:4	///	**Reserved**
3	Middle of Cycle	**Middle of Cycle** 1 = only the first half of a two-part bus cycle has completed. This condition can happen if the processor is stopped on the first bus cycle of a 16-bit access to an odd address. Do not to insert any bus cycles when this bit is set; the processor must first be single-stepped one bus cycle to compete the access before any bus cycles are forced by the JTAG debugger.
2	OP_FETCH	**OP_FETCH** 1 = processor is expecting to fetch an opcode. This lets the debugging software know that the processor is about to fetch an opcode. Software can then substitute its own opcodes in place of the ones from the system to interrogate or change the value of any register.
1	Breakpoint Flag	**Breakpoint Flag** 1 = the breakpoint is currently being searched. It is set when the ENB_BKPT instruction is loaded into the instruction register. 0 = breakpoint has been found.
0	JTAG HALT	**JTAG HALT** 1 = target CPU is halted.

Hardware Breakpoints

There are four hardware BREAKPOINT scan chains. Each BREAKPOINT chain is identical to the DATA scan chain described above except with some additional features in the CTL field and an 8 bit COUNT field is added. The hardware breakpoint chains can be used as four individual breakpoints, or they can be used in pairs for specifying either address ranges or in a value/mask pair.

The control byte is as follows:

- Bits [3:0] of the control byte must match for the desired cycle to trigger. If CTL[2:0]=111 then the cycle type field is ignored and a trigger will occur on any type of cycle.
- The hardware breakpoints can be used in pairs for masking or to specify ranges. HW_BKPT[1] can be paired with HW_PKPT[2] and HW_BKPT[3] can be paired with HW_BKPT[4].
- If HW_BKPT[1 or 3] is programmed with CTL[7:6]=00, the trigger condition in HW_BKPT[1] must be met; then HW_BKPT[2] is enabled and must also be met before the processor is halted. Only the ADDR is compared in this mode.
- If HW_BKPT[1 or 3] is programmed with CTL[7:6]=01, the value on the ADDR, DATA and STATUS busses are logically ANDed with value in HW_BKPT[2 or 4] and the result is compared with value in HW_BKPT[1 or 3]. The CTL byte in HW_BKPT[2 or 4] is ignored when used as a pair.
- If HW_BKPT[1 or 3] is programmed with a 10, the breakpoints are used in pairs to specify an address range as follows:
- HW_BKPT[1].ADDR >= Current ADDR >= HW_BKPT[2].ADDR

- If CTL[7:6]=11, the breakpoint is a simple address and cycle type comparison. The cycle type can be optionally enabled with the low 3 bits of the CTL register. This mode simply compares the current address on the bus with the value in the ADDR field of the HW_BKPT. The DATA field is ignored. If CTL[2:0] is not equal to 11, the desired cycle type must also match before the breakpoint triggers. Note that this mode compares the value on the address bus and triggers on instruction queue fills.
- If CTL[5]=1, the breakpoint is set to Opcode Execute Breakpoint Mode. In this mode, the ADDR field is compared with the segment register and instruction pointer and a breakpoint is initiated when the ADDR matches the physical address pointed to by the segment register and instruction pointer. The comparison is NOT on the ADDR bus, but directly on the execution units Instruction Pointer. This stops the CPU typically on the bus cycle when the opcode is about to be executed. This mode is ideal for breaking on specific opcodes as it will not break on the prefetch of an opcode. In this mode CTL[7:6] are ignored.
- If CTL[4]=1, the breakpoint is in Trace Qualifier mode. In this mode, the value in the ADDR field, or the ADDR and DATA values in a register pair is used as a qualifier to store data into the TRACE buffer.
- The COUNT field counts the number of occurrences that the trigger condition must be met before the processor is halted.
  0 = disables the breakpoint.
  1 = halts on the first occurrence.
  0xff = halts on the 255th occurrence.
- To enable a breakpoint, the desired breakpoint condition is scanned into the HW_BKPT scan chain. Then the BREAKPT_ENB instruction should be executed and the target processor is then allowed to run by issuing a RUN instruction. To stop the processor, deassert control signals to the CPU after the breakpoint condition is recognized. The breakpoint cannot stop on the actual breakpoint condition due to pipeline restrictions. Consequently, the processor stops at an instruction boundary.
- When the breakpoint is in Opcode Execute Mode, the processor stops before the instruction is executed. In the other breakpoint modes, the processor is stopped on the next instruction boundary after the desired bus cycle has been detected.

Trace Buffer

The TRACE scan chain is a 48-bit chain that provides access to the 256x48 bit trace buffer in the CPUDICE core. The TRACE scan chain is also identical to the DATA scan chain. Each word in the TRACE buffer corresponds to one bus cycle.

The TRACE buffer operates in three different modes, Normal TRACE mode, Branch history mode, and Timer mode. In normal TRACE mode, each word in the TRACE buffer corresponds to a single bus cycle. Trace mode can optionally use the four hardware breakpoints as qualifiers for the data to be stored into the TRACE buffer.

In Branch History mode, the 48-bit bus is split into two 24-bit words. The low 24-bit word corresponds to the physical address pointed to by the segment register and Instruction Pointer value before the branch was taken (the SOURCE address). The high 24-bit word corresponds to the CS:IP value after the branch is taken (the TARGET address). The NEWIP signal from the CPU core is active high when the IP is about to be reloaded. The IP bus on the CPU is the Instruction Pointer value from within the CPU execution unit. The IP bus is latched while NEWIP is active to capture the SOURCE address. The clock after NEWIP is active the IP bus will be latched again to capture the TARGET address.

Note: *The SOURCE address always points to the last byte of the opcode. Therefore, software must read the SOURCE address, then disassemble the code at that address and look a few bytes backwards to find the instruction that caused the branch. If an interrupt was taken, then TARGET address indicates that an interrupt was taken and, as a result, the SOURCE address will point to the address of the return address after the interrupt has been serviced.*

Timer mode also splits the 48 bit TRACE buffer into two 24 bit words. In this mode, each 24 bit word corresponds to the number of clock cycles divided by 8 between triggers. The most significant 2 bits indicate which trigger caused a store to the TRACE buffer. If the counter overflows the 22-bit count range, the maximum value of all ones is stored.

The current value of the TRACE buffer pointer is available in the upper eight bits of the TRACE_CTL scan chain. You can set the current address by scanning in a value into this register. After a trigger has occurred, the register reflects the current pointer in the TRACE buffer. Each time the TRACE scan chain is read, the pointer automatically increments to the next value. Consequently, if the trace buffer size is 256, reading the TRACE scan chain 256 times reads all values in the buffer. The value scanned into the TRACE scan chain is also written into that location of the TRACE buffer. This lets you initialize the entire buffer to make it easier to verify that the TRACE buffer contains valid data.

Typically, the TRACE_CTL scan chain is set to 0x00 and the TRACE scan chain is read N times (where N is the size of the trace buffer), while 48 bits of zeros are shifted in during the read. This initializes the entire TRACE buffer with zeroes. The desired trace mode is then selected via the TRACE_CTL and the HW_BKPT scan chains. After a trigger occurs, the TRACE buffer is read N times. The first value read is the oldest value stored in the buffer, the second value read is the next oldest and the last value read is the most recent value. Any values that are all zeroes are probably unused values.

Table 4-3. TRACE Buffer

Bits	Field Name	Description
15:8	TRACE Buffer Address	**TRACE Buffer Address**
7:6	///	**Reserved**
5	TRACE SIZE	**TRACE SIZE (Read Only)** 1 = 256 words. 0 = 16 words.
4	Trace Buffer Test Mode	**Trace Buffer Test Mode** 1 = no writes to trace buffer allowed when bus cycles execute. 0 = writes to trace buffer allowed when bus cycles execute.
3:2	Mode	**Mode** Mode 00 = normal 01 = branch history 10 = timer mode 11 = reserved
1:0	///	**Reserved**

DEICE Instructions

The Instruction register selects certain modes of operation and scan chains as described in Table 4-4. All instructions require the least-significant two bits to be 01, which require the INST register to be set to the appropriate settings per the 1149.1 specification.

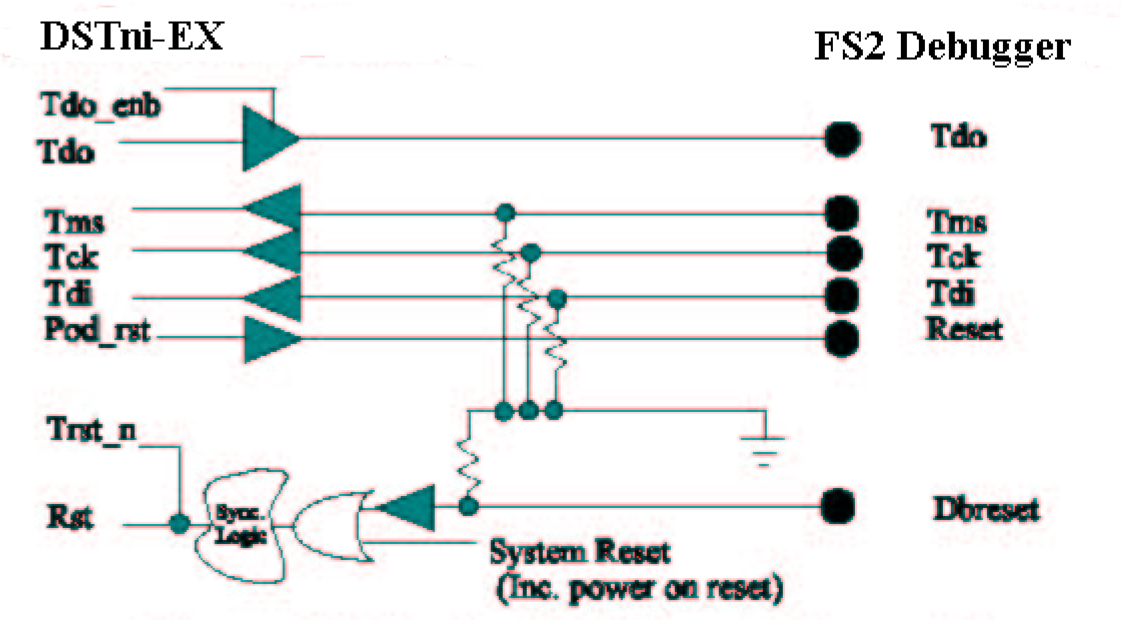
Table 4-4. DEICE Instructions

Instruction	Hex	Scan Chain	Description
EXTEST	00		Required by IEEE 1149.1
STOP	11	ADDR	Forces READY low. CPU stops execution immediately. Remains halted until the RUN instruction is executed.
RUN	21	ADDR	Releases the processor from HALT.
ENB_BKPT	31	STATUS	Run until the breakpoint is reached
STATUS	41	STATUS	Select the STATUS scan chain
ONE_OP	51	ADDR	Execute one opcode
ONE_CLK	61	ADDR	Execute one processor bus cycle
OP_ADDR	71	ADDR	Capture the address of the next opcode fetch
ADDR	81	ADDR	Select the ADDR scan chain
DATA	91	DATA	Select the DATA scan chain
TRACE	A1	TRACE	Select the TRACE scan chain
TRACE_CTL	B1	TR_CTL	Select the TRACE Control scan chain
BREAKPT1	C1	BREAKPT1	Select BREAKPOINT 1 scan chain
BREAKPT2	D1	BREAKPT2	Select BREAKPOINT 2 scan chain
BREAKPT3	E1	BREAKPT3	Select BREAKPOINT 3 scan chain
BREAKPT4	F1	BREAKPT4	Select BREAKPOINT 4 scan chain
MEM_WRITE	05	DATA	Initiates a MEMORY WRITE cycle
MEM_READ	15	DATA	Initiates a MEMORY READ cycle
IO_WRITE	25	DATA	Execute an IO WRITE cycle
IO_READ	35	DATA	Execute an IO READ Cycle
APPLY	45	DATA	Apply the scanned in vector for 1 bus cycle
APPLY_CPU	55	DATA	Apply the scanned in vector for 1 bus cycle only to the CPU. Hold WRN, WRLN, WRHN and RDN inactive to external logic.
APPLY_EXT	65	DATA	Apply the scanned in vector for 1 bus cycle only to external logic. The CPU remains inactive
RESET	75	DATA	Assert reset to the CPU and external peripherals
BYPASS	FF	BYPASS	This instruction required by IEEE 1149.1. The BYPASS opcode is automatically loaded into the INSTRUCTION register when RESET is asserted

FS2 Target Connection

Figure 4-1 shows a typical connection from the CPU core to the First Silicon Solutions Debugger.

Figure 4-1. Typical FS2 Target Connection



Excerpt From First Silicon Solution's VSA186 Debugger User's Guide

The standard target connection is the 20-position flat ribbon cable with the AMP System 50 connector. This mates to AMP connector 104549-2 (vertical surface mount), 104069-1 (right-angle through-hole), or 104068-1 (vertical through-hole) mounted on the target.

Pin	Signal	I/O	Active	Comments
1	///	///	///	The target should not connect to these pins.
2	DBRESET	OUT	HIGH	Driven HIGH by the debugger to reset the target system. Typically hooked into the target power-on reset circuit.
3	RESET	IN	HIGH	Input to debugger informs debugger that a target reset has occurred.
4	GND			Signal reference
5	///	///	///	The target should not connect to these pins.
6	VCC	IN	///	Used by debugger to determine target power-on state. Debugger does not draw significant current from this pin.
7	///	///	///	The target should not connect to these pins.
8	GND	///	///	Signal reference
9	///	///	///	The target should not connect to these pins.
10	GND	///	///	Signal reference
11	///	///	///	The target should not connect to these pins.
12	TDI	OUT	HIGH	JTAG signal
13	TDO	IN	HIGH	JTAG signal
14	TMS	OUT	HIGH	JTAG signal
15	GND	///	///	Signal reference
	///	///	///	
16	TCK	OUT	HIGH	JTAG signal
17	GND			Signal reference
18	TRST#	OUT	LOW	JTAG signal (optional)
19	DBINST#	OUT	LOW	Driven low by the debugger
20	BSEN#	OUT	LOW	Driven low by the debugger

Notes: *When designing in a target system connector for the debugger, pay close attention to the TCK signal. TCK is an edge-sensitive signal where ringing is undesirable.*

DBRESET can be active HIGH or LOW. Configurable by the FS2 debugger. The RSTIN# is active LOW. BSCEN must be HIGH for JTAG to work with the debugger.

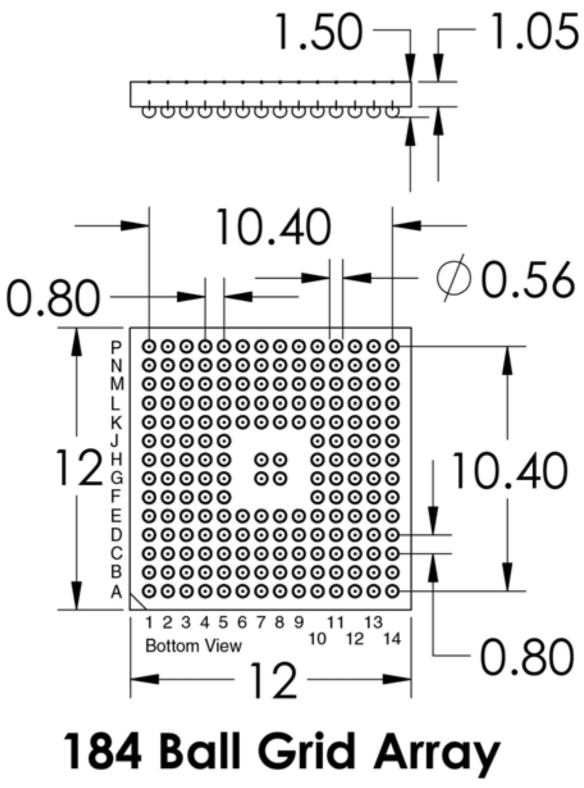
5: Packaging and Electrical

This chapter describes the DSTni packaging and electrical characteristics.

Packaging

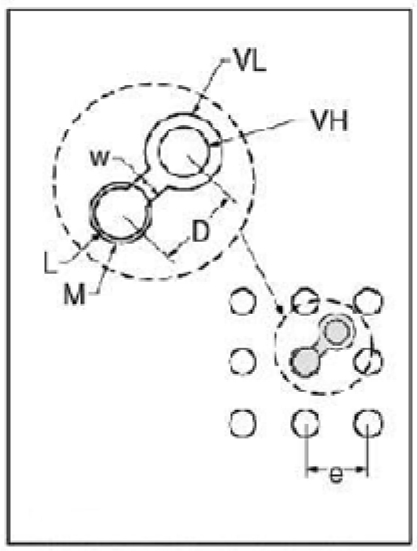
The DSTni-EX package is a 12-by-12 mm LFBGA with 0.8mm ball pitch. The part has four thermal balls in the center to increase heat dissipation. Die size is 4.1 x 5.4 mm in 0.18u TSMC process. Figure 5-1 describes the package.

Figure 5-1. DSTni Package



Recommended Circuit Board Layout

Figure 5-2. Recommended Circuit Board Layout



Recommended PCB Design Rules

Component Land Pad Diameter	0.35
Solder Land (L) Diameter	0.33
Opening in Solder Mask (M) Diameter	0.44
Solder (Ball) Land Pitch (e)	0.80
Line Width Between Via and Land (w)	0.13
Distance Between Via and Land (D)	0.56
Via Land (VL) Diameter	0.51
Through Hole (VH) Diameter	0.25
Pad ArrayMatrix or External Row	14 x 14
Periphery Rows	5

Note: 3x3 matrix shown for illustration only, one land pad shown with via connection. The component land pad diameter refers to the pad opening on the component side (solder mask defined). The package has solder balls in the center in addition to periphery rows of balls.

Electrical Specifications

Absolute Maximum Ratings

Table 5-1. Absolute Maximum Ratings

Parameter	Sym	Min	Max	Units
Core Supply Voltage	VDD1.8	-0.5	2.5	V
IO Supply Voltage	VDD3.3	-0.5	4.6	V
Input Voltage	Vi	-0.5	6	V
Output Voltage	Vo	-0.5	6	V
ESD Performance	>3K (HBM), 300 (MM), 1000 (CDM)			V
Latch-Up current	Ilatch	>500		mA
Operating Temperature	TOPT	-40	125	^{o}C
Storage Temperature	TSTG	-65	150	^{o}C
Thermal Resistance (Case)	(θ_{JC})		7.6	^{o}C/W
Thermal Resistance (Ambient)	(θ_{JA})		30	^{o}C/W
Package Dissipation	105Deg C Ta		0.67	W
Package Dissipation	95 Deg C Ta		1	W
Package Dissipation	85 Deg C Ta		1.33	W
Package Dissipation	70 Deg C Ta		1.83	W

***Note:** Long-term exposure to absolute maximum ratings may affect device reliability, and permanent damage may occur if operation exceeds the rating. The device should be operated under recommended operating conditions.*

Recommended Operating Conditions

Table 5-2. Recommended Operating Conditions

Parameter	Min	Typ	Max	Unit
Core Supply Voltage (10%)	1.62	1.8	1.98	V
IO Supply Voltage (10%)	3.0	3.3	3.6	V
T_J Junction Temperature	-40	25	125	°C
VIL Input Low Voltage	-0.3	///	0.8	V
VIH Input High Voltage	2.0	///	5.5	V
VT Threshold Point (non-Schmitt Input)	1.46	1.60	1.76	V
VT+ Schmitt trig low to high threshold point	1.50 wc	1.55	1.55 bc	V
VT- Schmitt trig high to low threshold point	0.88 wc	0.95	0.98 bc	V
II Input leakage current @Vi-3.3V or 0V	///	±10na	±1ua	///
IOZ Tri-state output leakage current @Vo-3.3V	///	10na	1ua	///
IOZ Tri-state output leakage current @Vo-0V	///	-10na	-1ua	///
RPU Pull-up Resistor	56K	77K	122K	Ω
RPD Pull-down Resistor	51K	69K	127K	Ω
Ball Input Capacitance	///	4	///	pf
VOL Output low voltage @IOL max	///	///	0.4	V
VOH Output high voltage @IOH max	2.4	///	///	V
IOL Low level output current @VOL 0.4V 2mA	2.2	3.5	4.3	mA
IOL Low level output current @VOL 0.4V 4mA	4.4	7.1	8.5	mA
IOL Low level output current @VOL 0.4V 8mA	8.8	14.1	17.0	mA
IOL Low level output current @VOL 0.4V 12mA	13.2	21.2	25.5	mA
IOL Low level output current @VOL 0.4V 16mA	17.6	28.2	34.0	mA
IOL Low level output current @VOL 0.4V 24mA	24.2	38.8	46.7	mA
IOH High level output current @VOH 2.4V 2mA	-3.2	-6.4	-10.0	mA
IOH High level output current @VOH 2.4V 4mA	-6.4	-12.8	-20.0	mA
IOH High level output current @VOH 2.4V 8mA	-12.8	-25.7	-40.0	mA
IOH High level output current @VOH 2.4V 12mA	-19.1	-38.5	-60.0	mA
IOH High level output current @VOH 2.4V 16mA	-28.7	-57.7	-90.0	mA
IOH High level output current @VOH 2.4V 24mA	-38.2	-76.9	-119.9	mA
Input Rise and fall time (10% <>90%)	///	///	8	ns
CPUCLK (0 wait internal RAM)	1	///	100	Mhz
CPUCLK (1 wait internal RAM)	1	///	115	Mhz
VDD1.8 Current (1Mhz)	///	15	20	mA
VDD1.8 Current (127Mhz)	///	150	200	mA
VDD3.3 Current	///	150	200	mA
PLL Jitter p/p over 200 cycles	///	48	200	ps
PLL Lock Time	20	///	150	us

Typical values are at 25°C and are for design information only and are not guaranteed and not production tested.

Note: *DSTni-EX uses two power supply voltages, one for core logic (1.8V) and another for I/O (3.3V). If the 3.3V supply is powered and the 1.8V core logic is not powered, current in excess of 350ma will flow into the chip. This is not a problem for short periods not to exceed 1 minute. Longer periods could overheat DSTni and cause device failure.*

I/O Characteristics – Xin/Xout Pins

Table 5-3. I/O Characteristics˜ Xin/Xout Pins

Parameter	Sym	Min	Typ	Max	Units
Input Clock Frequency Tolerance	Δf	///	///	±100	ppm
Input Clock Duty Cycle	TDC	35	///	65	%
Input Capacitance	CIN	///	3.0	///	pF

PHY Receiver Input Characteristics

Table 5-4. PHY Receiver Input Characteristics

Item	Spec	Units	Comments
Full Scale Input voltage	3.0 Differential pk-to-pk	V	
Input Common Mode	1.6-2.0	V	Gain dependent.

100Base-TX Transceiver Characteristics

Table 5-5. 100Base-TX Transceiver Characteristics

Parameter	Sym	Min	Typ	Max	Units
Peak Differential Output Voltage	VP	0.95	///	1.05	V
100M TX mid-level	///	-50	///	50	mV
Signal Amplitude Symmetry	VSS	98	///	102	%
Signal Rise/Fall Time	TRF	3.0	///	5.0	ns
Rise/Fall Time Symmetry	TRFS	///	///	0.5	ns
Duty Cycle Distortion	DCD	35	50	65	%
Overshoot/Undershoot	VOS	///	///	5	%
Jitter (measured differentially)	///	///	///	1.4	ns

100Base-FX Transceiver Characteristics

Table 5-6. 100Base-FX Transceiver Characteristics

Parameter	Sym	Min	Typ	Max	Unit	Test Conditions
Transmitter						
Peak Differential Output Voltage	VP	0.6	///	1.5	V	///
Signal Rise/Fall Time (2pF load)	TRF	///	///	1.9	ns	10<->90%
Jitter (measured differentially)	///	///	///	1.3	ns	///
Receiver						
Peak Differential Input Voltage	VIP	0.55	///	1.5	V	///
Common Mode Input Range	VCMIR	///	///	VDD-0.7	V	///

100Base-T Transceiver Characteristics

Table 5-7. 100Base-T Transceiver Characteristics

Parameter	Sym	Min	Typ	Max	Unit	Test Conditions
Transmitter						
Peak Differential Output Voltage	VOP	2.2	2.5	2.8	V	With Transformer, line replaced by 100Ω resistor
Transition Timing Jitter added by the MAU and PLS sections	///	0	2	11	ns	After line model specified by IEEE 802.3 for 10BASE-T MAU
Receiver						
Receive Input Voltage	ZIN	///	3.6	///	kΩ	///
Differential Squelch Threshold	VDS	300	420	585	mV	///

100Base-T Link Integrity Timing Characteristics

Table 5-8. 100Base-T Link Integrity Timing Characteristics

Parameter	Sym	Min	Typ	Max	Units	Test Conditions
Time Link Loss Receive	TLL	50	///	150	ms	///
Link Pulse	TLP	2	///	7	Link Pulses	///
Link Min Receive Timer	TLR Min	2	///	7	ms	///
Link Max Receive Timer	TLR Max	50	///	150	ms	///
Link Transmit Period	TLT	8	///	24	ms	///
Link Pulse Width	TLPW	60	///	150	ms	///

Power Curve Diagrams

Figure 5-3 shows the current for both power supplies of a typical part.

Figure 5-3. DSTni Current

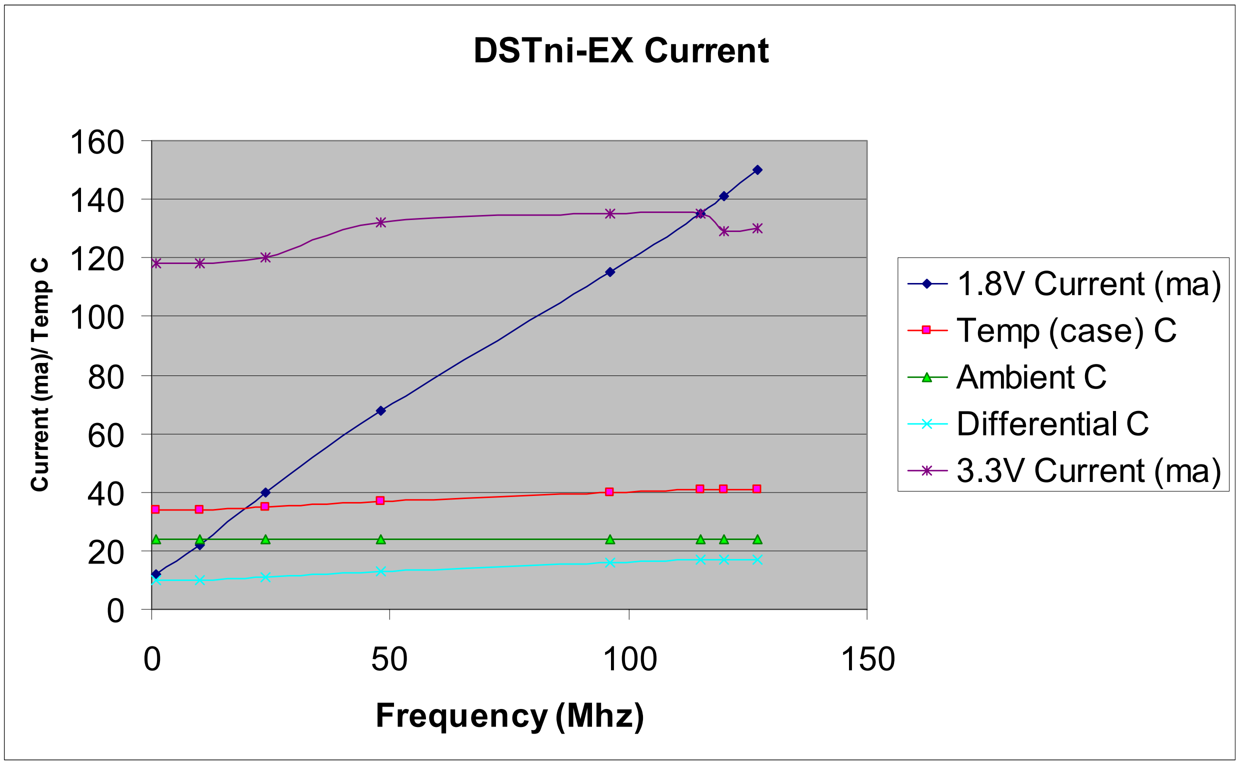
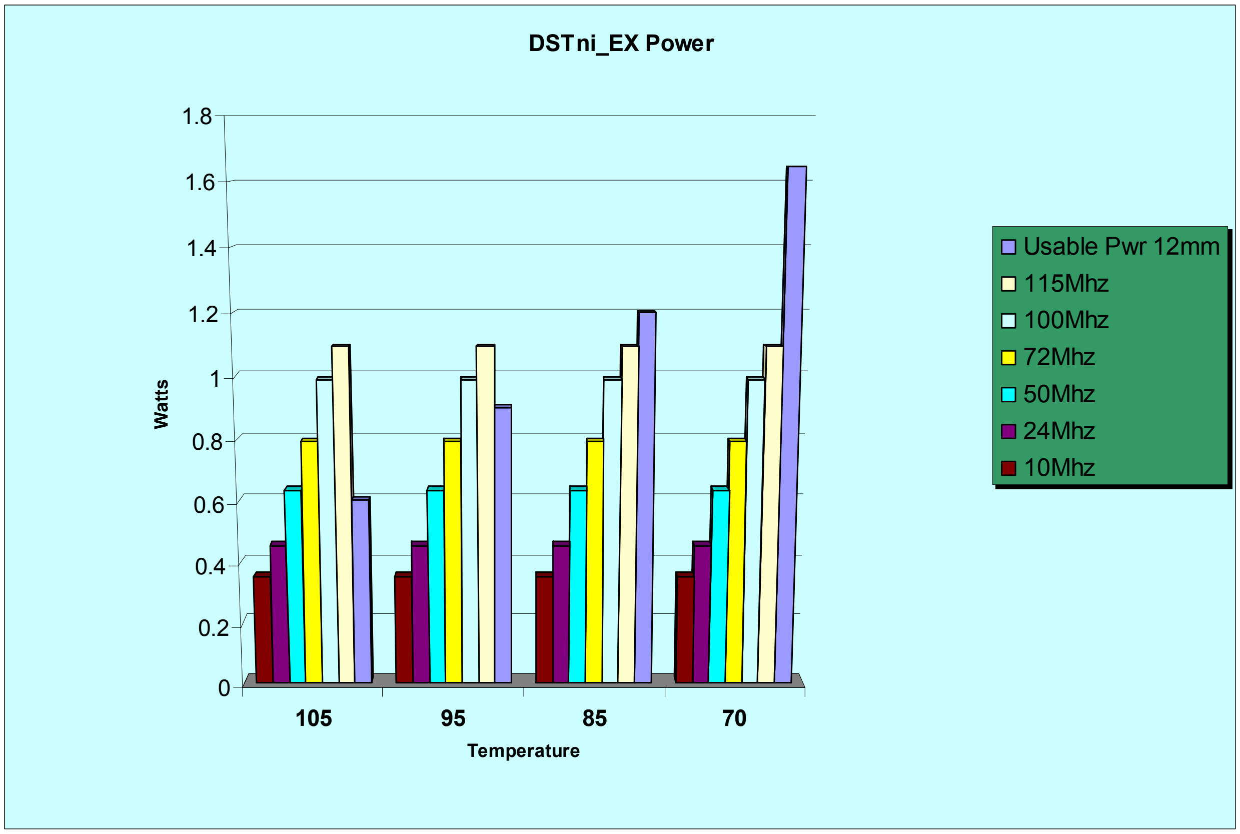


Figure 5-4 shows shows the power dissipation of a worse-case device at four different ambient temperatures.

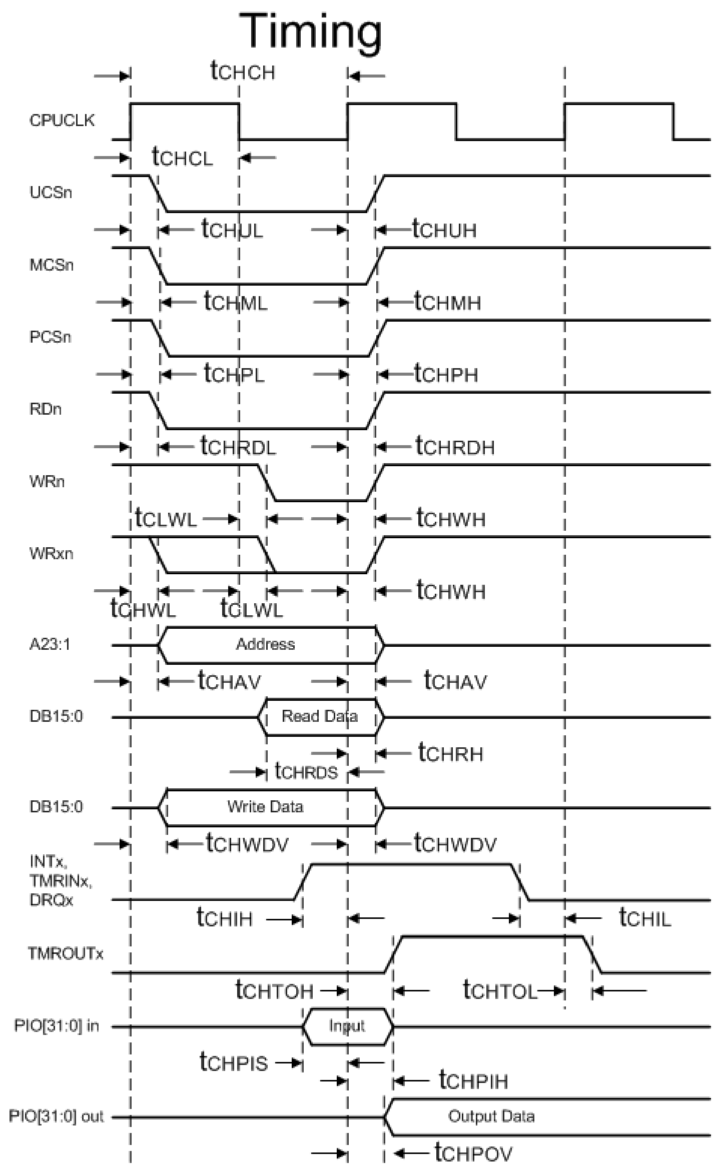
Figure 5-4. DSTni Power Curve



6: Applications

This appendix identifies various DSTni applications. Topics in this chapter include:

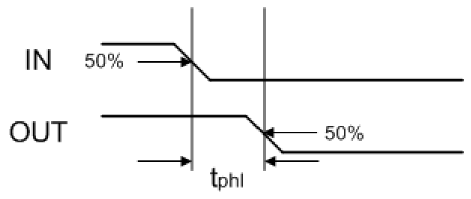
Timing



Note: All input signals are synchronized before use inside the DSTni-EX. All outputs use the rising edge of CPUCLK to generate the output signals. Any signals that use the falling edge of CPUCLK are shown on specific application diagrams. For slow digital signals hidden behind PIO pins or not shown above use tCHPIS for setup, tCHPIH for hold and tCHPOV for output delay.

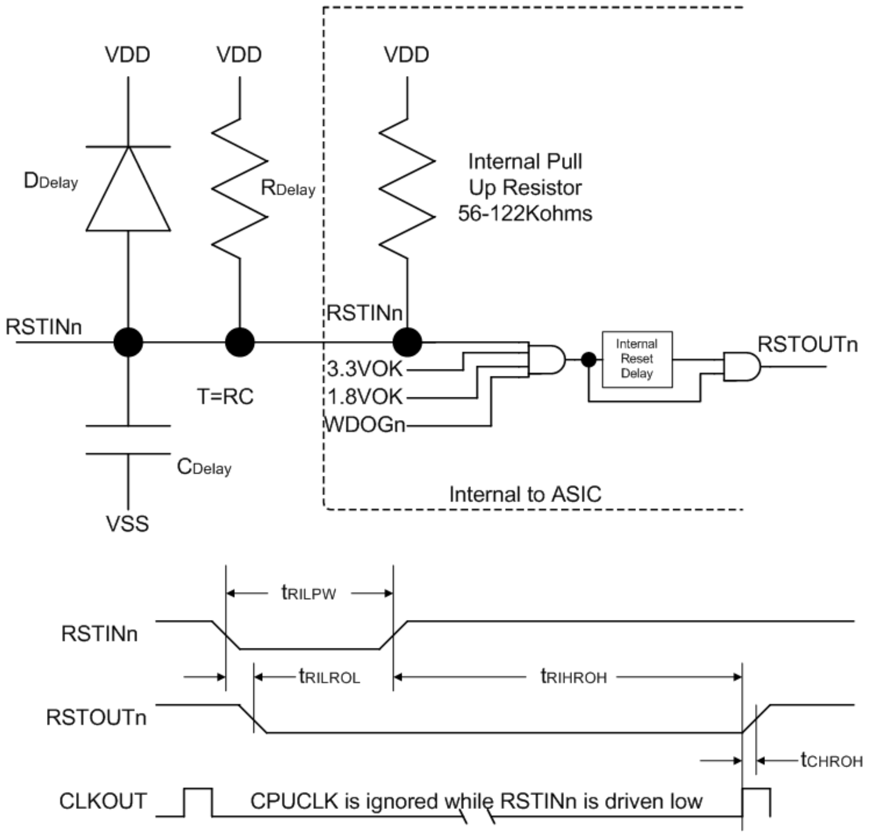
Data

Name	Description	Min	Typ	Max	units
Clock	CPU Clock Frequency	10.00		127.00	Mhz
tCHCL	Clock High to Clock Low	2.76		65.00	ns
tCHCH	Clock High to Clock High	7.87		100.00	ns
tCHUL	Clock High to UCSn Low	-0.25	1.38	3.00	ns
tCHUH	Clock High to UCSn High	0.00	1.75	3.50	ns
tCHPL	Clock High to PCSxn Low	0.00	1.38	2.75	ns
tCHPH	Clock High to PCSxn High	0.00	1.75	3.50	ns
tCHML	Clock High to MCSxn Low	-0.50	0.63	1.75	ns
tCHMH	Clock High to MCSxn High	-0.25	1.00	2.25	ns
tCHRDL	Clock High to RDn Low	-0.50	0.75	2.00	ns
tCHRDH	Clock High to RDn High	0.00	1.38	2.75	ns
tCLWL	Clock Low to WRxn Low	-1.50	-0.63	0.25	ns
tCHWH	Clock High to WRxn High	-1.00	0.00	1.00	ns
tCHWL	Clock High to WRxn Low	-1.50	-0.63	0.25	ns
tCHAV	Clock High to Address Valid	-1.00	0.00	1.00	ns
tCHRDS	Clock High to Read Data Setup	3.00	3.00	3.00	ns
tCHRH	Clock High to Read Data Hold	-0.50	-0.25	0.00	ns
tCHWDV	Clock High to Write Data Valid	-0.50	1.50	3.50	ns
tCLBCH	Clock Low to BCLK High	0.00	1.38	2.75	ns
tCHBCL	Clock High to BCLK Low	-0.50	0.75	2.00	ns
tCHBAL	Clock High to BAAn Low	-0.75	0.13	1.00	ns
tCHBAH	Clock High to BAAn High	-0.50	0.63	1.75	ns
tCHLBL	Clock High to LBAn Low	-0.75	0.25	1.25	ns
tCHLBH	Clock High to LBAn High	-0.25	0.75	1.75	ns
tRILROL	Reset In Low to Reset Out Low	3.50	8.50	13.50	ns
tRILPW	Reset In Low Pulse Width	3.50	11.00	18.50	ns
rRIHROH	Reset In High to Reset Out High	148.50	173.50	198.50	ms
tCHRIS	Clock High to Reset In Setup	3.00	3.00	3.00	ns
tCHROH	Clock High to Reset Out High	0.00	1.38	2.75	ns
tCHIS	Clock High to INTx, TMRINx, DRQx Setup	3.00	3.00	3.00	ns
tCHIH	Clock High to INTx, TMRINx, DRQx Hold	-0.50	-0.25	0.00	ns
tCHTOL	Clock High to Timer Out Low	0.00	1.50	3.00	ns
tCHTOH	Clock High to Timer Out High	0.00	1.75	3.50	ns
tCLARS	Clock Low to ARDY Setup	3.00	3.00	3.00	ns
tCLARH	Clock Low to ARDY Hold	-0.50	-0.25	0.00	ns
tCHPIS	Clock High to PIO IN Setup	3.00	3.00	3.00	ns
tCHPIH	Clock High to PIO IN Hold	-0.50	-0.25	0.00	ns
tCHPOV	Clock High to PIO Out Valid	0.00	1.75	3.50	ns
		🟦 Input Setup Times			
		🟨 Output Delay Times			
		🟪 Hold Times			



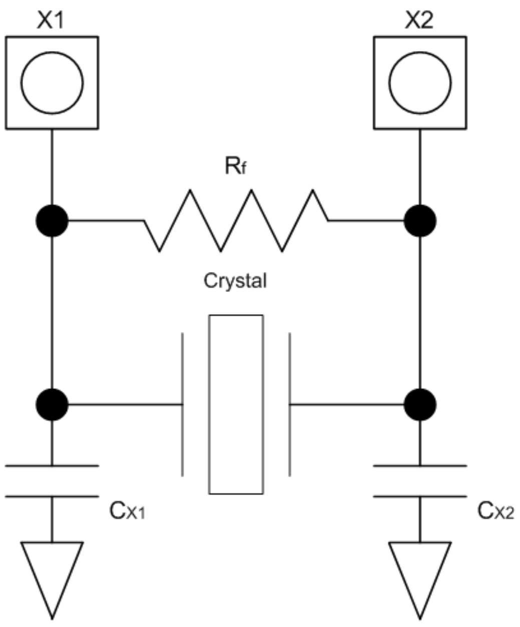
Note: *For non tri-state I/O cells, the propagation delay is measured from the 50% point of the input waveform to the 50% point of the output waveform. For tri-state I/O cells, since the tri-state status may not exhibit any change in the output waveform, we define the propagation delay (disable time) as the time form 50% of the disable signal to the turning on/off threshold level of the n and p MOS transistors.*

Reset



Power On Reset with no external connections is ~200ms. RSTOUTn is synchronized with the internal CLKOUT before it goes inactive. External components (Cdelay and Rdelay) can extend RSTOUTn if needed. The RSTINn input is LVTTL and TTL compatible.

XTAL

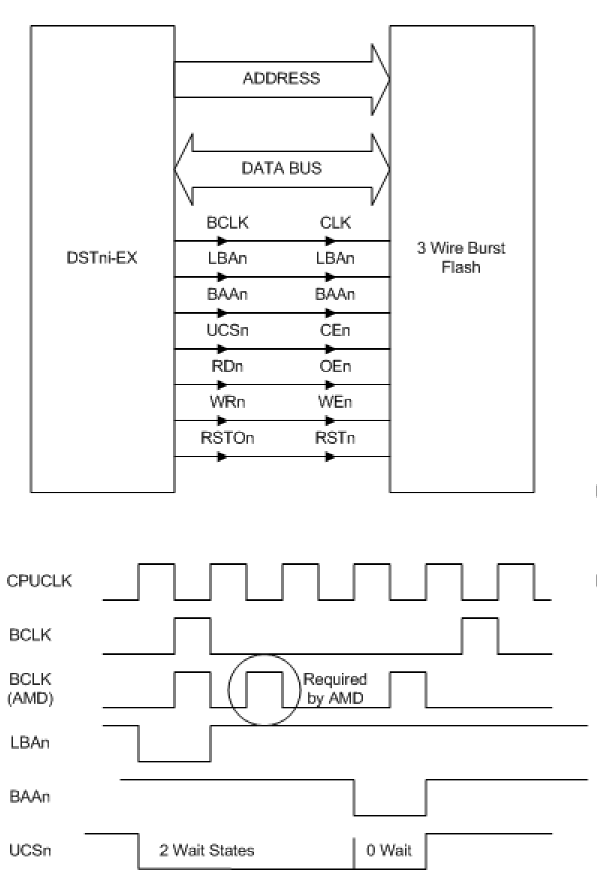


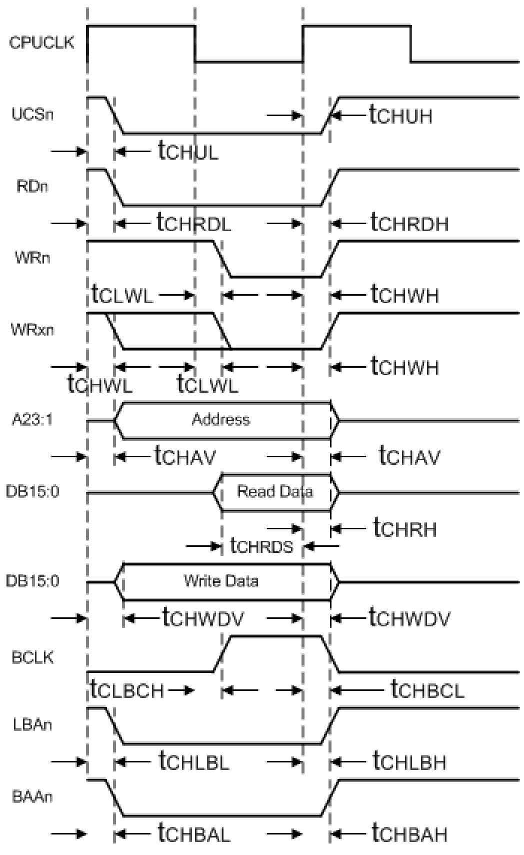
25Mhz Fundamental Mode Quartz Crystal (100PPM)	
Rf	1Mohm
Cx1	25pf
Cx2	25pf

25Mhz Fundamental Mode Ceramic Resonator (100PPM)	
Rf	1Mohm
Cx1	6pf
Cx2	6pf

Power On Oscillator Startup time is 50us maximum. DSTni requires this crystal to be 25 MHz to use either the internal PLL or Ethernet PHY. If using an external CPUCLK input (PLLBYP=0), this crystal still is required at 25 MHz to use the Ethernet PHY. If an external oscillator is used, it is connected to the X1 input and X2 is left open. CLKOUT cannot be used to drive X1. The capacitors Cx1 and Cx2 include PCB capacitance.

Burst Flash (3 wire)

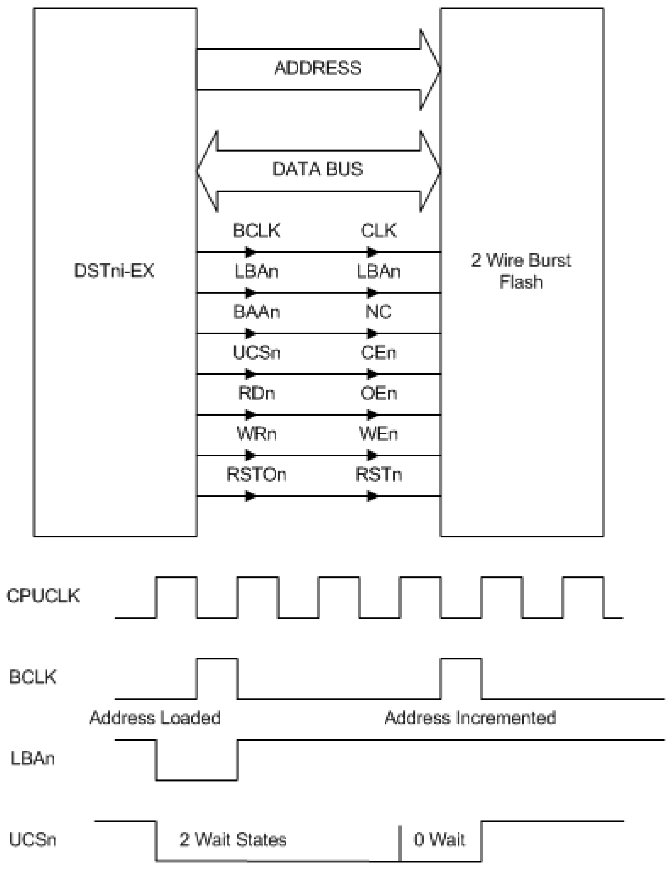


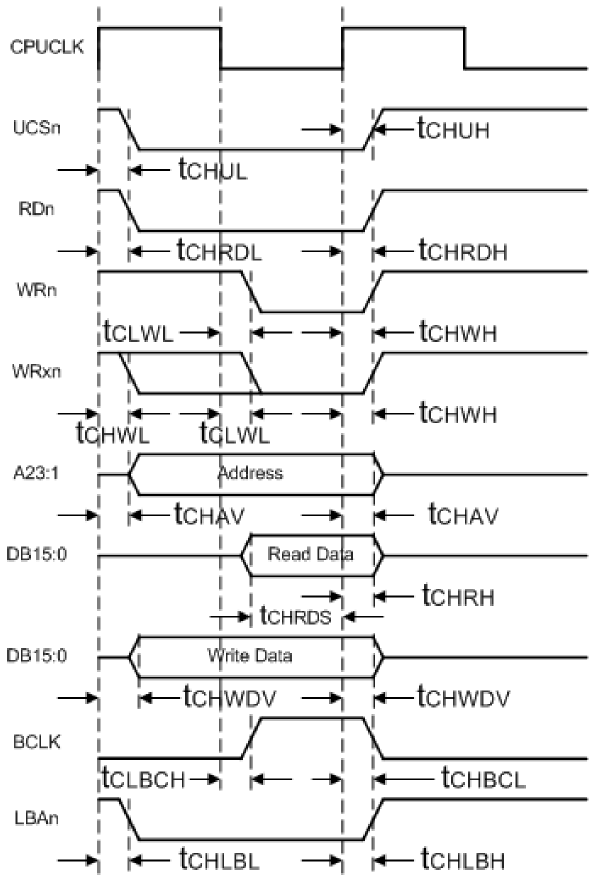


Flash access time is Initial: Flash Access + tCHCL + tCLBCH + tCHRDS.
Flash access time is Burst: Flash Access + tCHCL + tCLBCH + tCHRDS.
Example: 50Mhz clock (20ns) want to use 70ns flash (24ns burst)
Initial 70ns + 10ns + 2.75ns + 3ns = 85.75ns < (5 * 20ns) 4 wait states (#wait states+1*tCHCH)
Burst 24ns + 10ns + 2.75ns + 3ns = 38.75ns < (2 * 20ns) 1 wait states (#wait states+1*tCHCH)

Note: AMD burst flash devices require 2 clocks during an address load operation. Therefore connect CPUCLK to CLK on these devices.

Burst Flash (2 Wire)





Flash access time is Initial: Flash Access + tCHCL + tCLBCH + tCHRDS.

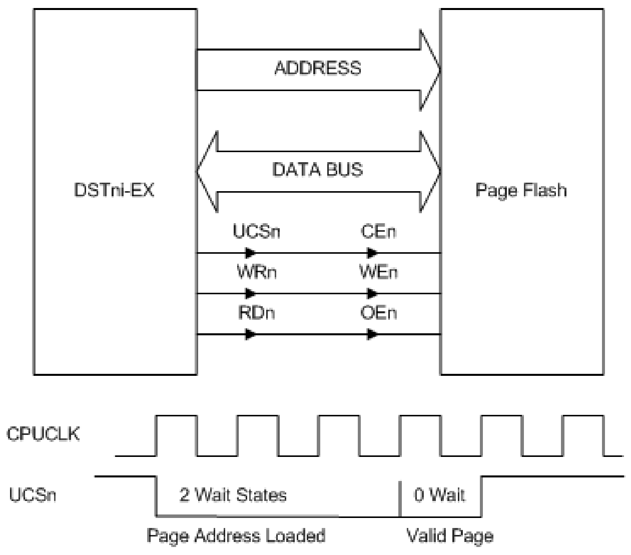
Flash access time is Burst: Flash Access + tCHCL + tCLBCH + tCHRDS.

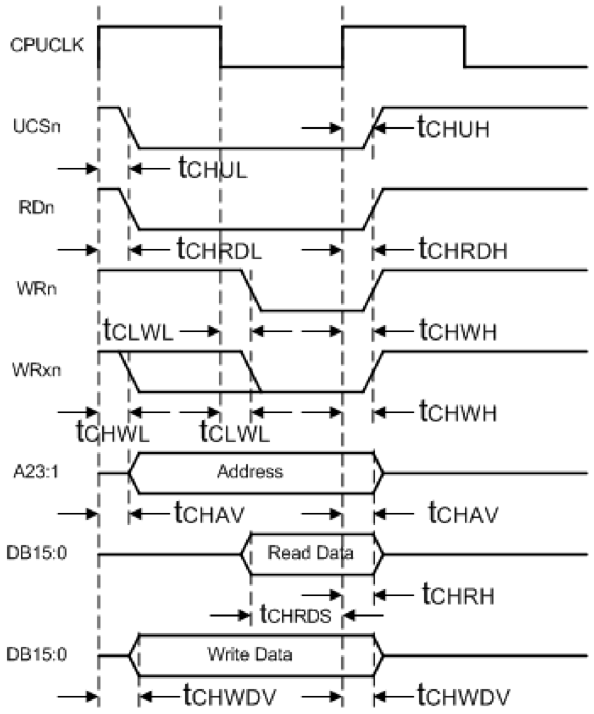
Example: 50Mhz clock (20ns) want to use 70ns flash (24ns burst)

Initial 70ns + 10ns + 2.75ns + 3ns = 85.75ns < (5 * 20ns) 4 wait states (#wait states+1*tCHCH)

Burst 24ns + 10ns + 2.75ns + 3ns = 38.75ns < (2 * 20ns) 1 wait states (#wait states+1*tCHCH)

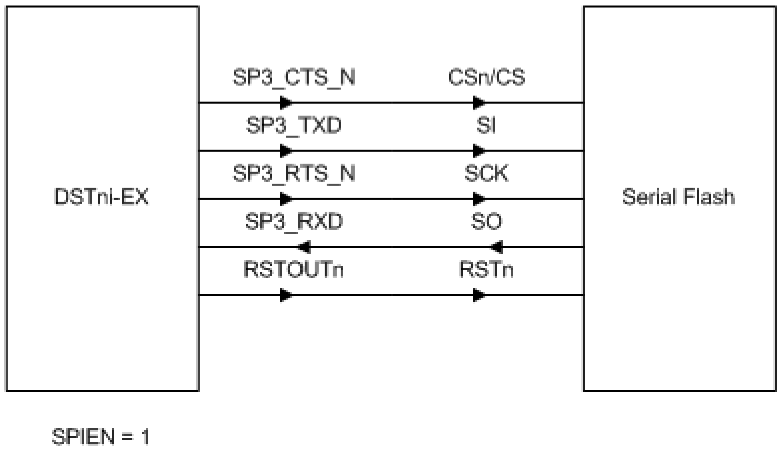
Page Flash

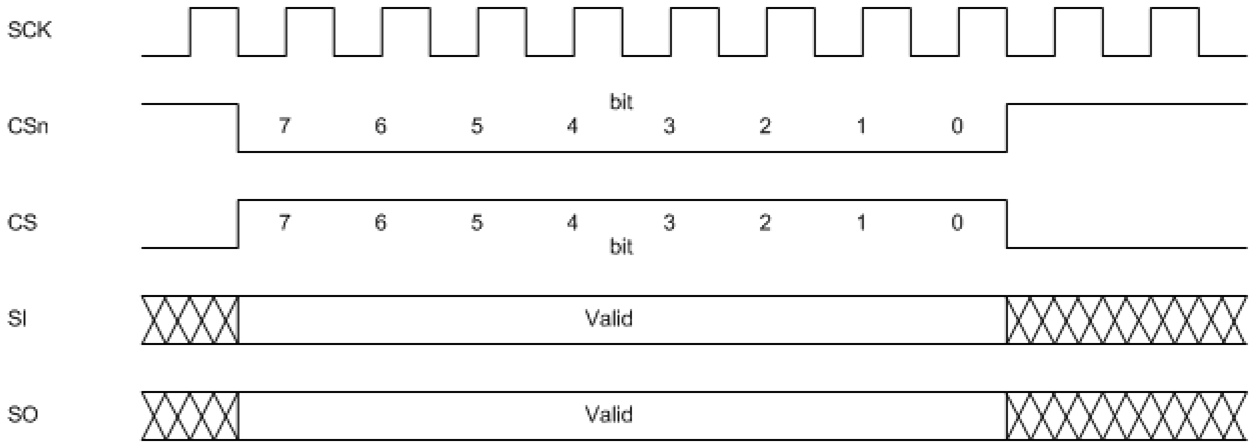




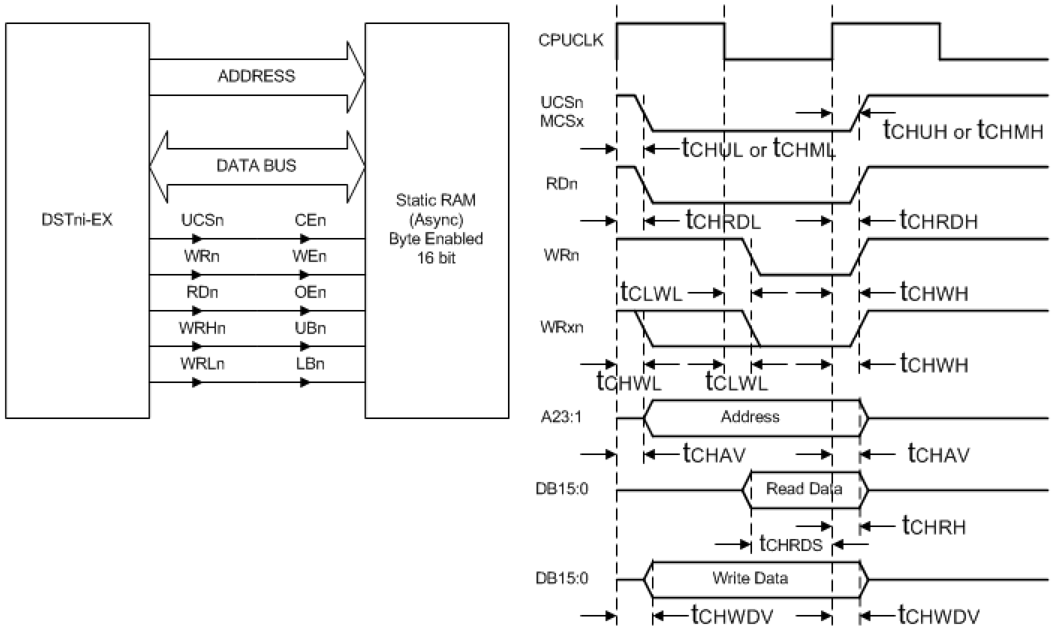
Flash access time is Initial: Flash Access + tCHAV + tCHRDS.
Flash access time is Burst: Flash Access + tCHAV + tCHRDS.
Example: 50Mhz clock (20ns) want to use 70ns flash (25ns page)
Initial 70ns + 1ns + 3ns = 74ns < (4 * 20ns) 3 wait states (#wait states+1*tCHCH)
Burst 25ns + 1ns + 3ns = 29ns < (2 * 20ns) 1 wait states (#wait states+1*tCHCH)

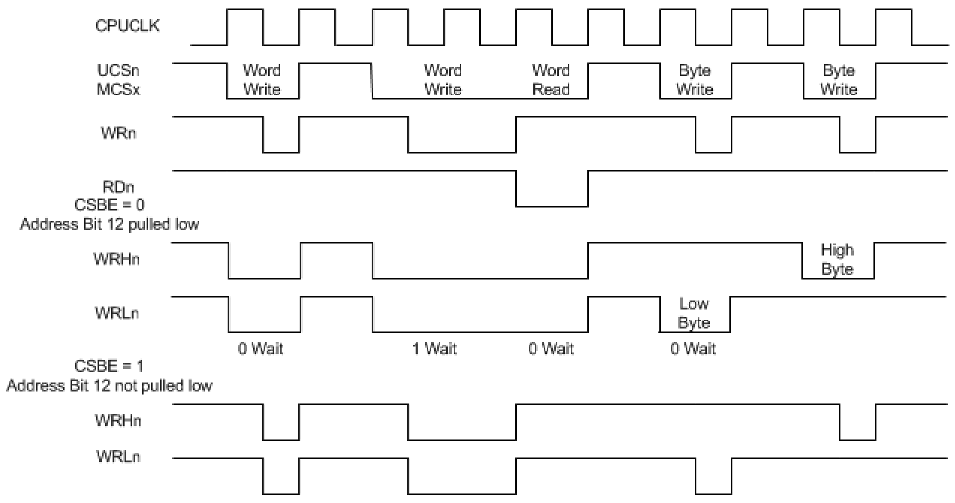
Serial Flash





Static RAM

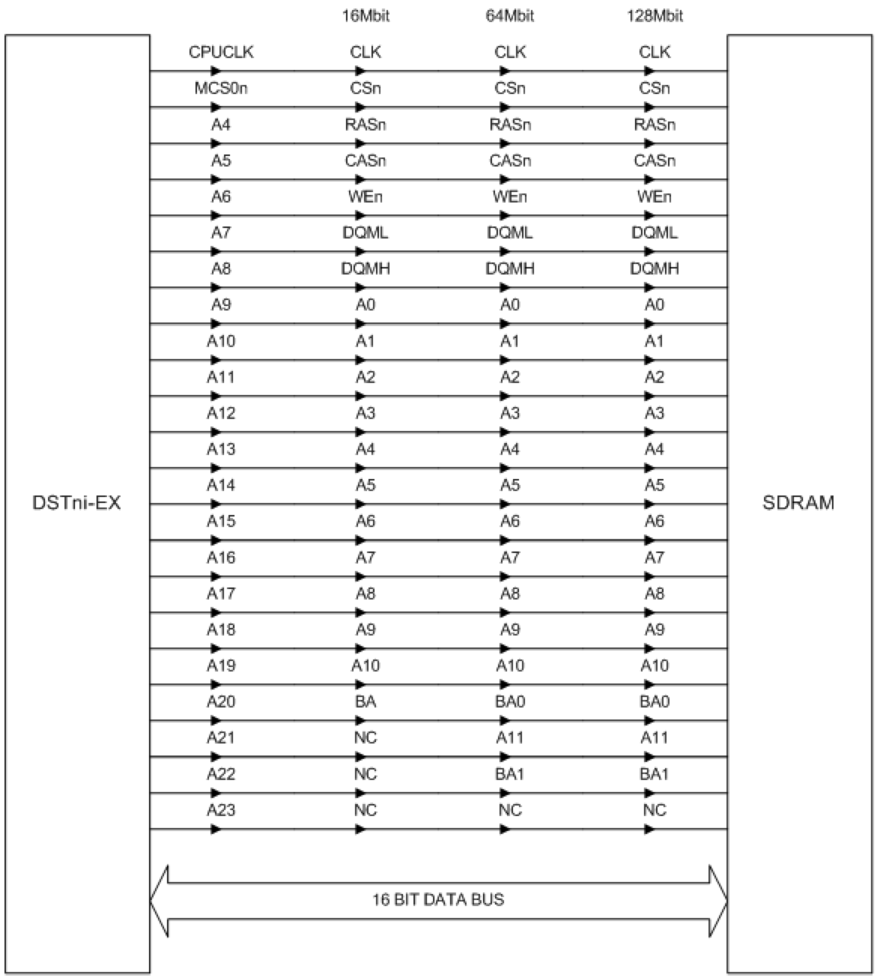




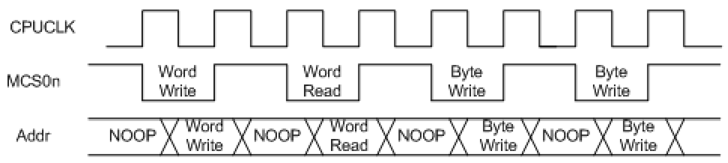
SRAM access time is: SRAM Access + tCHAV + tCHRDS.
Example: 50Mhz clock (20ns) want to use 45ns SRAM
45ns + 1ns + 3ns = 49ns < (3 * 20ns) 2 wait states (#wait states+1*tCHCH)

Note: By design the WRxn signals always return high a minimum 0.5ns before the address or data bus changes.

SDRAM

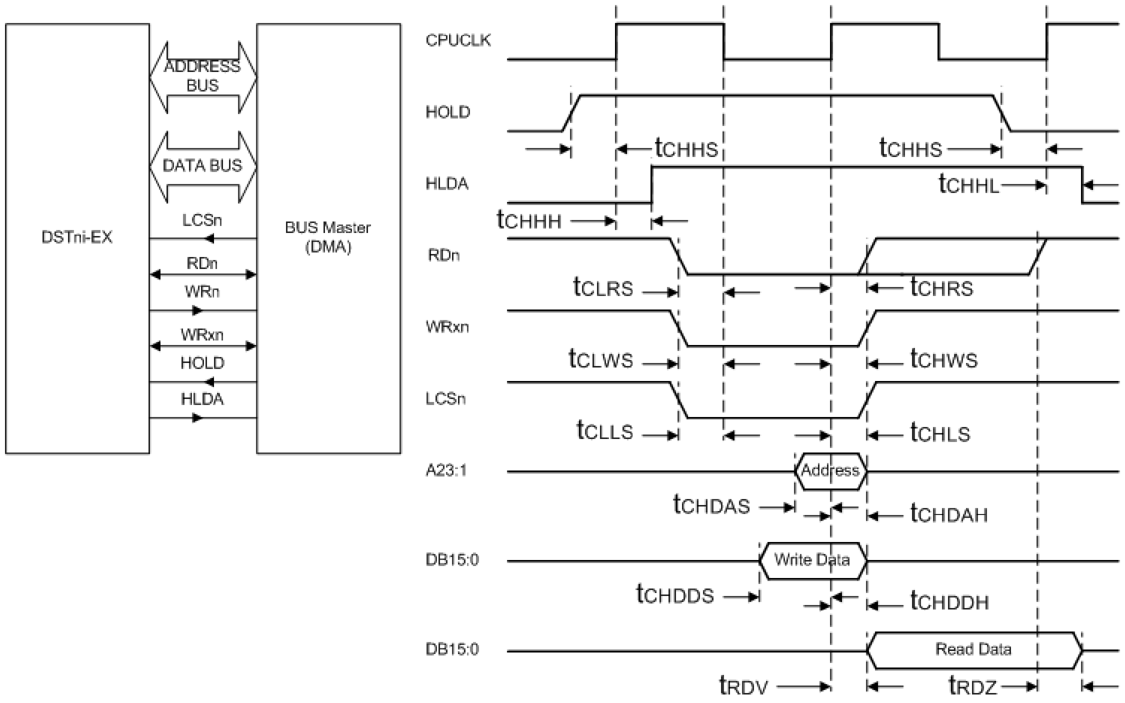


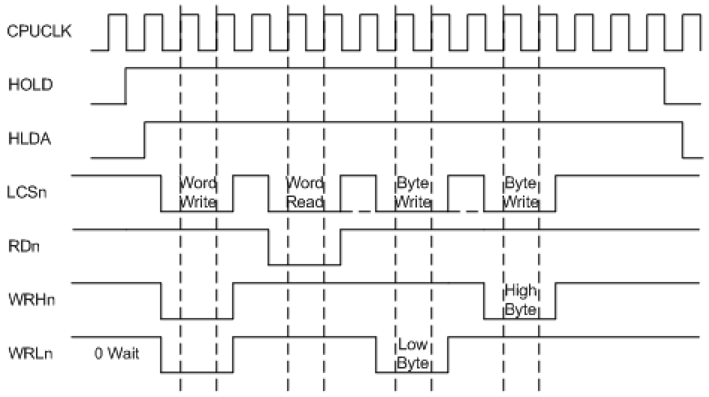
CKE should be pulled high



All SDRAM commands are passed through the Address lines.
Delays between SDRAM memory cycles are dependent on internal wait states and page hits.

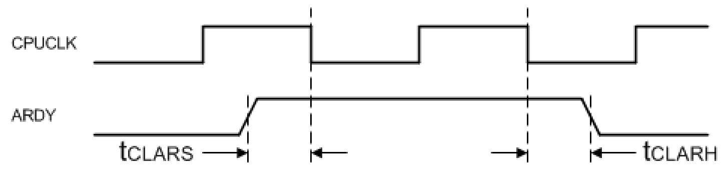
External DMA

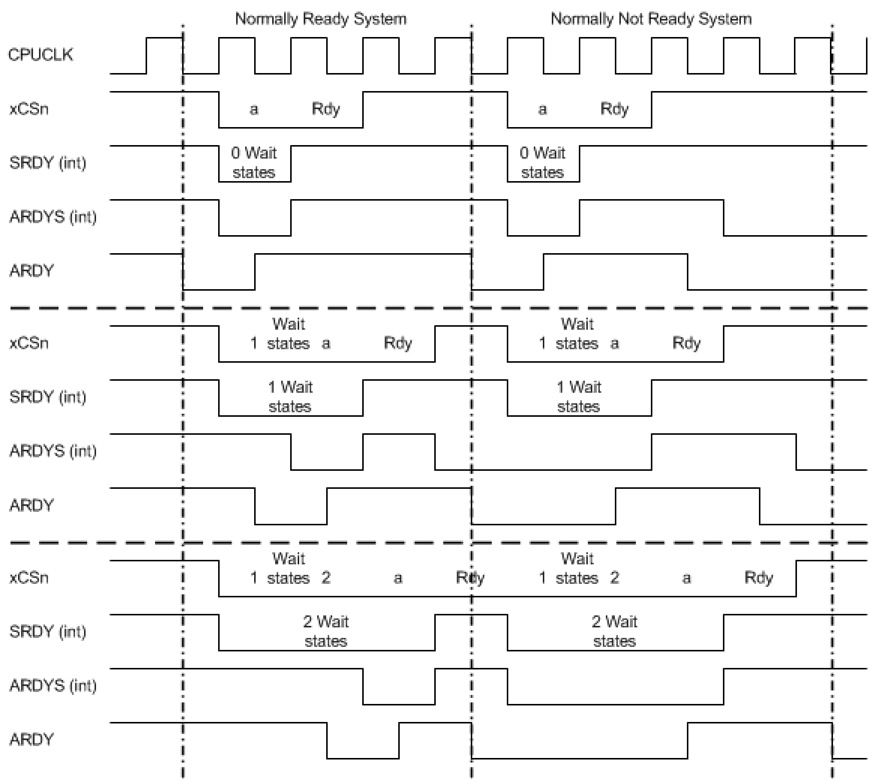




Note: *WRn is not used as an input for the external DMA interface. Wait states for the internal memory are not used for external DMA. The RDn and WRxn signals are leading-edge level sensitive. Memory reads and writes always complete internally in one clock cycle following the leading-edge detection. Therefore, the address bus is ignored after the rising edge of CPUCLK after the first leading edge of RDn and WRxn. The external DMA can hold the RDn signal LOW as long as needed for data to be read properly. The RDn can be released asynchronous to the CPUCLK. For reads, the data is latched from the memory after one clock and continues to be driven out the DATA BUS until released by RDn going HIGH. The external DMA can hold the WRxn signal as long as necessary. However, the data is only written on the first rising edge of CPUCLK after the WRxn goes LOW.*

ARDY

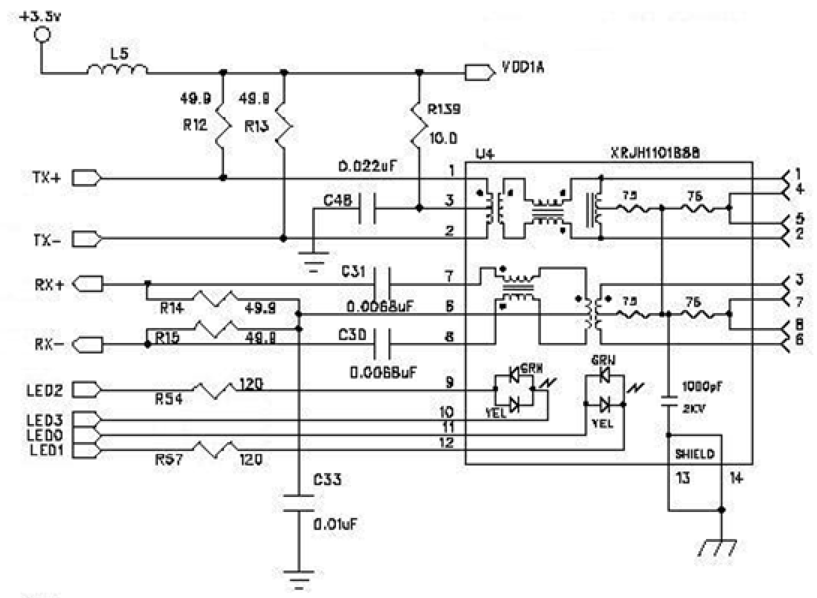


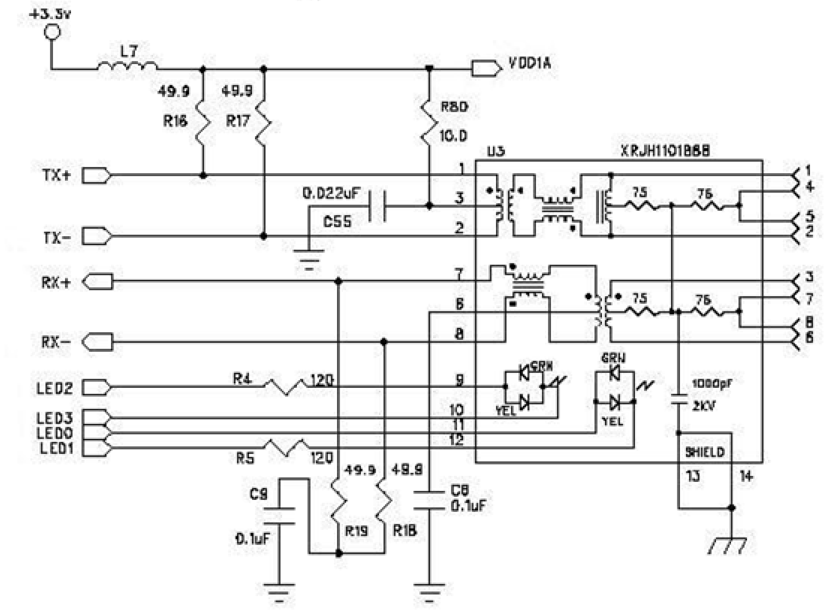


*<span>**Note:**</span> ARDY is internally synchronized on the falling edge of the CPUCLK. For a normally ready system and an access cycle set for 0, wait the peripheral cannot respond in time to cause the CPU to wait. If the system is normally not ready, this can be accomplished with difficulty. By adding 1 wait to the access cycle, the system can respond in time to cause the CPU to wait. However, less than ½ clock time is needed and can be difficult to generate on faster systems. By adding 2 waits, the system can easily respond in time to cause the CPU to wait in all systems.*

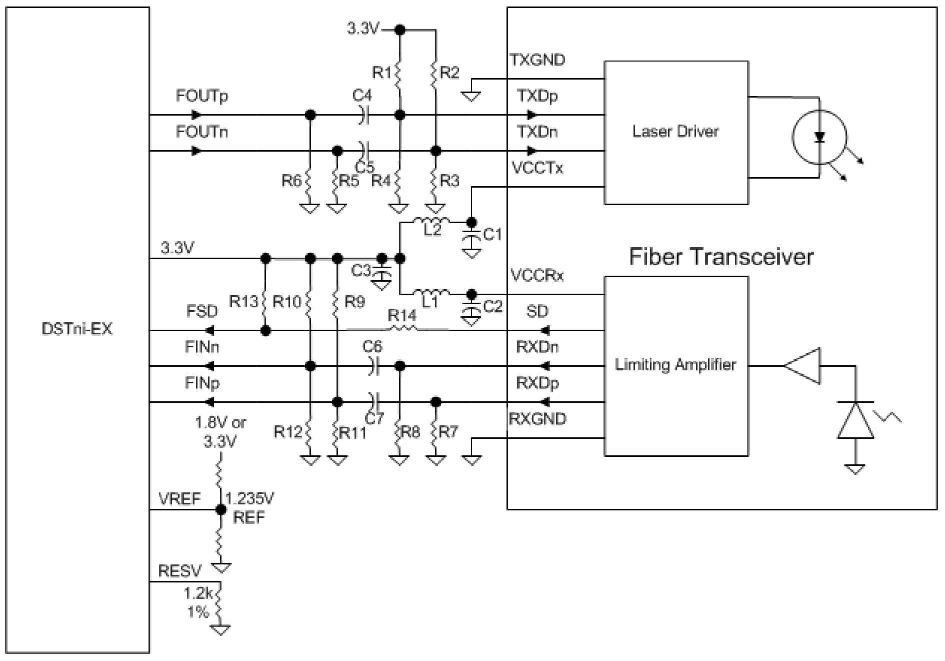
PHY (10/100 Mbit)

The following figure shows two different, but equivalent, ways to attach DSTni to an RJ-45 connection.





Fibre (100 Mbit)



C1/2/3	=4.7uF
C4/5/6/7	=10nF
L1 /2	=1uh
R1/2/9/10	=82ohm
R3/4/11/12	=130ohm
R7/8	=150ohm
R5/6	=50ohm

IF FIBER MODULE has PECL SD

R13	=10Kohm
R14	=open

IF FIBER MODULE has LVTTL SD

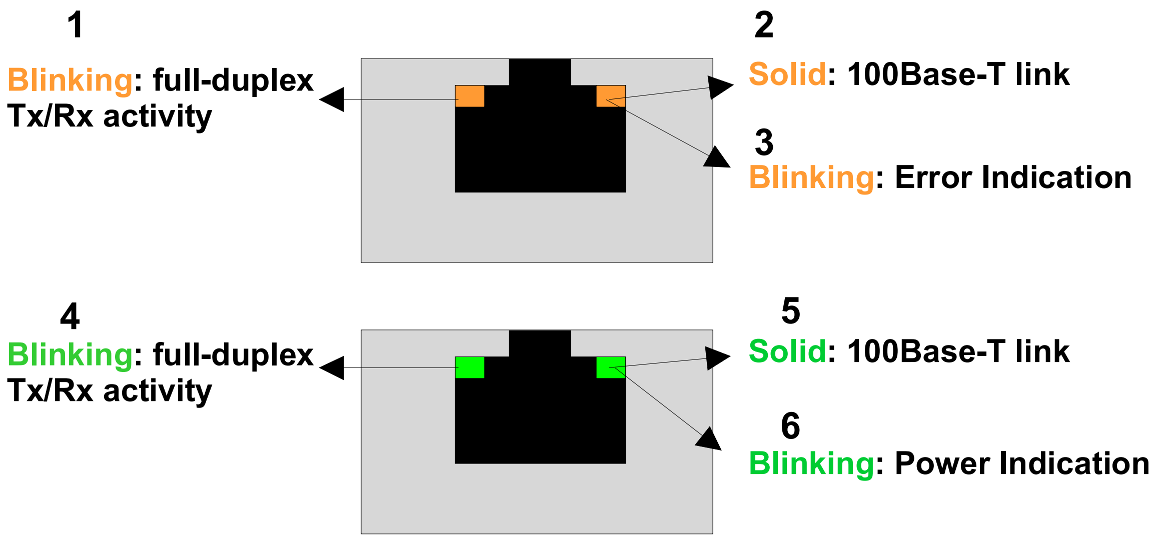
R13	=open
R14	=0 ohm

Note: *If Fiber is not used, leave all pins open.*

LED Functionality

This section shows the LEDs associated with DSTni and describes their functions.

Figure 6-1. LEDs


1
Blinking: full-duplex Tx/Rx activity
2
Solid: 100Base-T link
3
Blinking: Error Indication
4
Blinking: full-duplex Tx/Rx activity
5
Solid: 100Base-T link
6
Blinking: Power Indication

1. Blinks when transmitting or receiving packets with full-duplex LAN connection.

2. Solid when the link is established with a 100Base-T connection.

3. Blinks to indicate error detection. A pause equal to two blinking intervals is required between the x times. If the link is good, the LED color for the pause interval is yellow or green, depending on the 10 or 100M link. If the link is not good, the LED is OFF during the pause interval.
   Blinks 1 time = hardware error.
   Blinks 2 times = duplicated IP address on the network.
   Blinks 3 times = faulty network connection.
   Blinks 4 times = no DHCP response received.

4. Blinks when transmitting or receiving packets with half-duplex LAN connection.

5. Solid when the link is established with a 10Base-T connection.

6. Blinks when powered on. When a 10Base-T link is detected, the LED changes to solid green.

7: Instruction Clocks

Mnemonic	Opcode	Description	186	486	DSTni
AAA	37	ASCII-adjust AL after addition	8	3	4
AAD	D5 0A	ASCII-adjust AX before division	15	14	5
AAM	D4 0A	ASCII-adjust AL after multiplication	19	15	5
AAS	3F	ASCII-adjust AL after subtraction	7	3	4
ADC AL,imm8	14 ib	Add immediate byte to AL with carry	3	1	1
ADC AX,imm16	15 iw	Add immediate word to AX with carry	4	1	1
ADC r/m8,imm8	80 /2 ib	Add immediate byte to r/m byte with carry	4/16	1/3	1/4
ADC r/m16,imm16	81 /2 iw	Add immediate word to r/m byte with carry	4/16	1/3	1/4
ADC r/m16,imm8	83 /2 ib	Add sign-extended immediate byte to r/m word with carry	4/16	1/3	1/4
ADC r/m8,r8	10 /r	Add byte register to r/m byte with carry	3/10	1/1	1/4
ADC r/m16,r16	11 /r	Add word register to r/m word with carry	3/10	1/1	1/4
ADC r8,r/m8	12 /r	Add r/m byte to byte register with carry	3/10	½	1/4
ADC r16,r/m16	13 /r	Add r/m word to word register with carry	3/10	½	1/4
ADD AL,imm8	04 ib	Add immediate byte to AL	3	1	1
ADD AX,imm16	05 iw	Add immediate word to AX	4	1	1
ADD r/m8,imm8	80 /0 ib	Add immediate byte to r/m byte	4/16	1/3	1/4
ADD r/m16,imm16	81 /0 iw	Add immediate word to r/m word	4/16	1/3	1/4
ADD r/m16,imm8	83 /0 ib	Add sign-extended immediate byte to r/m word	4/16	1/3	1/4
ADD r/m8,r8	00 /r	Add byte register to r/m byte	3/10	1/1	1/4
ADD r/m16,r16	01 /r	Add word register to r/m word	3/10	1/1	1/4
ADD r8,r/m8	02 /r	Add r/m byte to byte register	3/10	½	1/4
ADD r16,r/m16	03 /r	Add r/m word to word register	3/10	½	1/4
AND AL,imm8	24 ib	AND immediate byte with AL	3	1	1
AND AX,imm16	25 iw	AND immediate word with AX	4	1	1
AND r/m8,imm8	80 /4 ib	AND immediate byte with r/m byte	4/16	1/3	1/4
AND r/m16,imm16	81 /4 iw	AND immediate word with r/m word	4/16	1/3	1/4
AND	83 /4 ib	AND sign-extended immediate byte	4/16	1/3	1/4

Mnemonic	Opcode	Description	186	486	DSTni
r/m16,imm8		with r/m word			
AND r/m8,r8	20 /r	AND byte register with r/m byte	3/10	1/1	1/4
AND r/m16,r16	21 /r	AND word register with r/m word	3/10	1/1	1/4
AND r8,m8	22 /r	AND r/m byte with byte register	3/10	½	1/4
AND r16,r/m16	23 /r	AND r/m word with word register	3/10	½	1/4
BOUND r16,m16&16	62 /r	Check to see if word register is within bounds	33-35	7,50	24
CALL rel16	E8 cw	Call near, displacement relative to next instruction	15	3	8
CALL r/m16	FF /2	Call near, register indirect/memory indirect	13/19	5/5	3+J (8)
CALL ptr16:16	9A cd	Call far to full address given	23	18	6+J (11)
CALL m16:16	FF /3	Call far to address at m16:16 word	38	17	7+J (12)
CBW	98	Put signed extension of AL in AX	2	3	1
CLC	F8	Clear Carry Flag	2	2	1
CLD	FC	Clear Direction Flag so the Source Index (SI) and/or the Destination Index (DI) registers will increment during instructions	2	2	1
CLI	FA	Clear Interrupt Enable Flag	2	5	1
CMC	F5	Complement Carry Flag	2	2	1
CMP AL,imm8	3C ib	Compare immediate byte to AL	3	1	1
CMP AX,imm16	3D iw	Compare immediate word to AX	4	1	1
CMP r/m8,imm8	80 /7 ib	Compare immediate byte to r/m byte	3/10	½	1/4
CMP r/m16,imm16	81 /7 iw	Compare immediate word to r/m word	3/10	½	1/4
CMP r/m16,imm8	83 /7 ib	Compare sign-extended immediate byte to r/m word	3/10	½	1/4
CMP r/m8,r8	38 /r	Compare byte register to r/m byte	3/10	½	1/4
CMP r/m16,r16	39 /r	Compare word register to r/m word	3/10	½	1/4
CMP r8,r/m8	3A /r	Compare r/m byte to byte register	3/10	½	1/4
CMP r16,r/m16	3B /r	Compare r/m word to word register	3/10	½	1/4
CMPS m8,m8	A6	Compare byte ES:[DI] to byte segment:[SI]	22	8	5
CMPS m16,m16	A7	Compare word ES:[DI] to word segment:[SI]	22	8	5
CMPSB	A6	Compare byte ES:[DI] to byte DS:[SI]	22	8	5
CMPSW	A7	Compare word ES:[DI] to word DS:[SI]	22	8	5
CWD	99	Put signed extension of AX in DX::AX	4	3	1
DAA	27	Decimal-adjust AL after addition	4	2	2
DAS	2F	Decimal-adjust AL after subtraction	4	2	2
DEC	FE /1	Subtract 1 from r/m byte	3/15	1/3	1/4

Mnemonic	Opcode	Description	186	486	DSTni
r/m8					
DEC r/m16	FF /1	Subtract 1 from r/m word	3/15	1/3	1/4
DEC r16	48 + rw	Subtract 1 from word register	3	1	1
DIV r/m8	F6 /6	AL=AX/(r/m byte); AH=remainder	29/35	16/16	13/16+I
DIV r/m16	F7 /6	AX=DX::AX/(r/m word); DX=remainder	38/44	24/24	21/24+I
ENTER imm16,imm8	C8 iw ib	Create stack frame for nested procedure	22+ 16(n-1)	14	9+ 6(n-1)
ENTER imm16,0	C8 iw 00	Create stack frame for non-nested procedure	15	14	7
ENTER imm16,1	C8 iw 01	Create stack frame for nested procedure	25	17	9+6(n-1)
ESC m	D8 /0	Takes trap 7	?	?	2+I
	D9 /1	Takes trap 7	?	?	2+I
	DA /2	Takes trap 7	?	?	2+I
	DB /3	Takes trap 7	?	?	2+I
	DC /4	Takes trap 7	?	?	2+I
	DD /5	Takes trap 7	?	?	2+I
	DE /6	Takes trap 7	?	?	2+I
	DF /7	Takes trap 7	?	?	2+I
HLT	F4	Suspend instruction execution	2	4	1
IDIV r/m8	F6 /7	AL=AX/(r/m byte); AH=remainder	44-52/ 50-58	19/20	15/18+I
IDIV r/m16	F7 /7	AX=DX::AX/(r/m word); DX=remainder	53-61/ 59-67	27/28	23/26+I
IMUL r/m8	F6 /5	AX=(r/m byte)*AL	25-28/ 31-34	5/5	12/15
IMUL r/m16	F7 /5	DX::AX=(r/m word) *AX	34-37/ 40-43	5/6	20/23
IMUL r16,r/m16,imm 8	6B /r ib	(word register)=(r/m word)*(sign-extended byte integer)	22-25	5/5	20/23
IMUL r16,r/m16,imm 16	69 /r iw	(word register)=(r/m word)*(sign-extended word integer)	29-32	5/6	20/23
IN AL,imm8	E4 ib	Input byte from immediate port to AL	10	17	6
IN AX,imm8	E5 ib	Input word from immediate port to AX	10	17	6
IN AL,DX	EC	Input byte from port in DX to AL	8	17	4
IN AX,DX	ED	Input word from port in DX to AX	8	17	4
INC r/m8	FE /0	Increment r/m byte by 1	3/15	1/3	1/4
INC r/m16	FF /0	Increment r/m word by 1	3/15	1/3	1/4
INC r16	40 + rw	Increment word register by 1	3	1	1
INS m8,DX	6C	Input byte from port in DX to ES:[DI]	14	15	3
INS m16,DX	6D	Input word from port in DX to ES:[DI]	14	15	3
INSB	6C	Input byte from port in DX to ES:[DI]	14	15	3

Mnemonic	Opcode	Description	186	486	DSTni
INSW	6D	Input word from port in DX to ES:[DI]	14	15	3
INT 3	CC	Generate interrupt 3 (trap to debugger)	45	26	2+I
INT imm8	CD ib	Generate type of interrupt specified by immediate byte	47	30	2+I
INTO	CE	Generate interrupt 4 if Overflow Flag (OF) is 1	48,4	28/3	2+I
IRET	CF	Return from interrupt handler to interrupted procedure	28	15	7
JA rel8	77 cb	Jump short if above	13,4	3/1	2+J
JAE rel8	73 cb	Jump short if above or equal	13,4	3/1	2+J
JB rel8	72 cb	Jump short if below	13,4	3/1	2+J
JBE rel8	76 cb	Jump short if above or equal	13,4	3/1	2+J
JC rel8	72 cb	Jump short if carry	13,4	3/1	3+J
JCXZ rel8	E3 cb	Jump short if above	15,5	8/5	2+J
JE rel8	74 cb	Jump short if equal	13,4	3/1	2+J
JG rel8	7F cb	Jump short if greater	13,4	3/1	2+J
JGE rel8	7D cb	Jump short if greater or equal	13,4	3/1	2+J
JL rel8	7C cb	Jump short if less	13,4	3/1	2+J
JLE rel8	7E cb	Jump short if less or equal	13,4	3/1	2+J
JMP rel8	EB cb	Jump short direct, displacement relative to next instruction	14	3	2+J
JMP rel16	E9 cw	Jump near direct, displacement relative to next instruction	14	3	2+J
JMP r/m16	FF /4	Jump near indirect, displacement relative to next instruction	11/17	5	2+J
JMP ptr16:16	EA cd	Jump far direct to doubleword immediate address	14	17	2+J
JMP m16:16	FF /5	Jump m16:16 indirect and far	26	13	2+J
JNA rel8	76 cb	Jump short if not above	13,4	3/1	2+J
JNAE rel8	72 cb	Jump short if not above or equal	13,4	3/1	2+J
JNB rel8	73 cb	Jump short if not below	13,4	3/1	2+J
JNBE rel8	77 cb	Jump short if not below or equal	13,4	3/1	2+J
JNC rel8	73 cb	Jump short if not carry	13,4	3/1	2+J
JNE rel8	75 cb	Jump short if not equal	13,4	3/1	2+J
JNG rel8	7E cb	Jump short if not greater	13,4	3/1	2+J
JNGE rel8	7C cb	Jump short if not greater or equal	13,4	3/1	2+J
JNL rel8	7D cb	Jump short if not less	13,4	3/1	2+J
JNLE rel8	7F cb	Jump short if not less or equal	13,4	3/1	2+J
JNO rel8	71 cb	Jump short if not overflow	13,4	3/1	2+J
JNP rel8	7B cb	Jump short if not parity	13,4	3/1	2+J
JNS rel8	79 cb	Jump short if not sign	13,4	3/1	2+J
JNZ rel8	75 cb	Jump short if not zero	13,4	3/1	2+J
JO rel8	70 cb	Jump short if overflow	13,4	3/1	2+J
JP rel8	7A cb	Jump short if parity	13,4	3/1	2+J
JPE rel8	7A cb	Jump short if parity even	13,4	3/1	2+J
JPO rel8	7B cb	Jump short if parity odd	13,4	3/1	2+J
JS rel8	78 cb	Jump short if sign	13,4	3/1	2+J
JZ rel8	74 cb	Jump short if zero	13,4	3/1	2+J
LAHF	9F	Load AH with low byte of Processor Status Flags register	2	3	1
LDS r16,m16:16	C5 /r	Load DS:r16 with segment:offset from memory	18	6/12	8
LEA r16,m16	8D /r	Load offset for m16 word in 16-bit register	6	1 / 2	2
LEAVE	C9	Destroy procedure stack frame	8	5	5

Mnemonic	Opcode	Description	186	486	DSTni
LES	C4 /r	Load ES:r16 with segment:offset from memory	18	6/12	8
LOCK	F0	Asserts LOCK during an instruction execution	1	1	0
LODS m8	AC	Load byte segment:[SI] in AL	12	5	4
LODS m16	AD	Load word segment:[SI] in AX	12	5	4
LODSB	AC	Load byte segment:[SI] in AL	12	5	4
LODSW	AD	Load word segment:[SI] in AX	12	5	4
LOOP rel8	E2	Decrement count; jump short if CX /=0	16,6	7/6	3+J
LOOPE rel8	E1 cb	Decrement count; jump short if CX /=0 and ZF=1	16,6	9/6	3+J
LOOPNE	E0 cb	Decrement count; jump short if CX /=0 and ZF=0	16,6	9/6	3+J
LOOPNZ	E0 cb	Decrement count; jump short if CX /=0 and ZF=0	16,6	9/6	3+J
LOOPZ	E1 cb	Decrement count; jump short if CX /=0 and ZF=1	16,6	9/6	3+J
MOV r/m8,r8	88 /r	Copy register to r/m byte	2	1	1
MOV r/m16,r16	89 /r	Copy register to r/m word	12	1	1
MOV r8,r/m8	8A /r	Copy r/m byte to register	2	1	1/4
MOV r16,r/m16	8B /r	Copy r/m word to register	9	1	1/4
MOV r/m16,sreg	8C /sr	Copy segment register to r/m word	2/11	3	1
MOV sreg,r/m16	8E /sr	Copy r/m word to segment register	2/9	3/9	1/4
MOV AL,moffs8	A0	Copy byte at segment:offset to AL	8	1	1
MOV AX,moffs16	A1	Copy word at segment:offset to AX	8	1	1
MOV moffs8,AL	A2	Copy AL to byte at segment:offset	9	1	1
MOV moffs16,AX	A3	Copy AX to word at segment:offset	9	1	1
MOV r8,imm8	B0+rb	Copy immediate byte to register	3	1	1
MOV r16,imm16	B8+rw	Copy immediate word to register	3	1	1
MOV r/m8,imm8	C6 /0	Copy immediate byte to r/m byte	12	1	1
MOV r/m16,imm16	C7 /0	Copy immediate word to r/m word	12	1	1
MOVS m8,m8	A4	Copy byte segment:[SI] to ES:[DI]	14	7	3
MOVS m16,m16	A5	Copy word segment:[SI] to ES:[DI]	14	7	3
MOVSB	A4	Copy byte segment:[SI] to ES:[DI]	14	7	3
MOVSW	A5	Copy word segment:[SI] to ES:[DI]	14	7	3
MUL r/m8	F6 /4	AX=(r/m byte)*AL	26-28/ 32-34	5/5	12/15
MUL r/m16	F7 /4	DX::AX=(r/m word)*AX	35-37/ 41-43	5/6	20/23
NEG r/m8	F6 /3	Perform a two's complement negation of r/m byte	3/10	1/3	1/4

Mnemonic	Opcode	Description	186	486	DSTni
NEG r/m16	F7 /3	Perform a two's complement negation of r/m word	3/10	1/3	1/4
NOP	90	Perform no operation	3	1	1
NOT r/m8	F6 /2	Complement each bit in r/m byte	3/10	1/3	1/4
NOT r/m16	F7 /2	Complement each bit in r/m word	3/10	1/3	1/4
OR AL,imm8	0C ib	OR immediate byte with AL	3	1	1
OR AX,imm16	0D iw	OR immediate word with AX	4	1	1
OR r/m8,imm8	80 /1 ib	OR immediate byte with r/m byte	4/16	1/3	1/4
OR r/m16,imm16	81 /1 iw	OR immediate word with r/m word	4/16	1/3	1/4
OR r/m16,imm8	83 /1 ib	OR immediate byte with r/m word	4/16	1/3	1/4
OR r/m8,r8	08 /r	OR byte with r/m byte	3/10	1/3	1/4
OR r/m16,r16	09 /r	OR word with r/m word	3/10	1/3	1/4
OR r8,r/m8	0A /r	OR r/m byte with byte register	3/10	1/3	1/4
OR r16,r/m16	0B /r	OR r/m word with word register	3/10	1/3	1/4
OUT imm8,AL	E6 ib	Output AL to immediate port	9	19	4
OUT imm8,AX	E7 ib	Output AX to immediate port	9	19	4
OUT DX,AL	EE	Output AL to port in DX	7	19	1
OUT DX,AX	EF	Output AX to port in DX	7	19	1
OUTS DX,m8	6E	Output byte DS:[SI] to port in DX	14	14	4
OUTS DX,m16	6F	Output word DS:[SI] to port in DX	14	14	4
OUTSB	6E	Output byte DS:[SI] to port in DX	14	14	4
OUTSW	6F	Output word DS:[SI] to port in DX	14	14	4
POP m16	8F /0	Pop to word of stack into memory word	20	5	5
POP r16	58+rw	Pop to word of stack into word register	10	1	5
POP DS	1F	Pop to word of stack into DS	8	4	5
POP ES	07	Pop to word of stack into ES	8	4	5
POP SS	17	Pop to word of stack into SS	8	4	5
POPA	61	Pop DI, SI, BP, BX, DX, CX and AX	51	9	14
POPF	9D	Pop top word of stack into Processor Status Flags register	8	9/6	5
PUSH m16	FF /6	Push memory word onto stack	16	4	2
PUSH r16	50+rw	Push register word onto stack	10	1	2
PUSH imm8	6A	Push sign-extended immediate byte onto stack	10	1	2
PUSH imm16	68	Push immediate word onto stack	10	1	2
PUSH CS	0E	Push CS onto stack	9	4	2
PUSH SS	16	Push SS onto stack	9	4	2
PUSH DS	1E	Push DS onto stack	9	4	2
PUSH ES	06	Push ES onto stack	9	4	2

Mnemonic	Opcode	Description	186	486	DSTni
PUSHA	60	Push AX, CX, DX, BX, original SP, BP, SI and DI	36	11	14
PUSHF	9C	Push Processor Status Flags register	9	4/3	2
RCL r/m8,1	D0 /2	Rotate 9 bits of CF and r/m byte left once	2/15	3 / 4	1
RCL r/m8,CL	D2 /2	Rotate 9 bits of CF and r/m byte left CL times	5+n/ 17+n	3 / 4	4+n
RCL r/m8,imm8	C0 /2 ib	Rotate 9 bits of CF and r/m byte left imm8 times	5+n/ 17+n	2 / 4	4+n
RCL r/m16,1	D1 /2	Rotate 17 bits of CF and r/m word left once	2/15	3 / 4	1
RCL r/m16,CL	D3 /2	Rotate 17 bits of CF and r/m word left CL times	5+n/ 17+n	3 / 4	4+n
RCL r/m16,imm8	C1 /2 ib	Rotate 17 bits of CF and r/m word left imm8 times	5+n/ 17+n	2 / 4	4+n
RCR r/m8,1	D0 /3	Rotate 9 bits of CF and r/m byte right once	2/15	3 / 4	1
RCR r/m8,CL	D2 /3	Rotate 9 bits of CF and r/m byte right CL times	5+n/ 17+n	3 / 4	4+n
RCR r/m8,imm8	C0 /3 ib	Rotate 9 bits of CF and r/m byte right imm8 times	5+n/ 17+n	2 / 4	4+n
RCR r/m16,1	D1 /3	Rotate 17 bits of CF and r/m word right once	2/15	3 / 4	1
RCR r/m16,CL	D3 /3	Rotate 17 bits of CF and r/m word right CL times	5+n/ 17+n	3 / 4	4+n
RCR r/m16,imm8	C1 /3 ib	Rotate 17 bits of CF and r/m word right imm8 times	5+n/ 17+n	2 / 4	4+n
REP INS m8,DX	F3 6C	Input CX bytes from port in DX to ES:[DI]	8+8n	19+11D	4+ (D+3)n
REP INS m16,DX	F3 6D	Input CX words from port in DX to ES:[DI]	8+8n	19+11D	4+ (D+3)n
REP LODS m8	F3 AC	Load CX bytes from segment:[SI] in AL	6+11n	5/ 7+4D	4+ (D+3)n
REP LODS m16	F3 AD	Load CX words from segment:[SI] in AX	6+11n	5/ 7+4D	4+ (D+3)n
REP MOVS m8,m8	F3 A4	Copy CX bytes from segment:[SI] in ES:[DI]	8+8n	5/ 12+3D	4+ (D+3)n
REP MOVS m16,m16	F3 A5	Copy CX words from segment:[SI] in ES:[DI]	8+8n	5/ 12+3D	4+ (D+3)n
REP OUTS DX,m8	F3 6E	Output CX bytes from DS:[SI] to port in DX	8+8n	20+8D	4+ (D+3)n
REP OUTS DX,m16	F3 6F	Output CX words from DS:[SI] to port in DX	8+8n	20+8D	4+ (D+3)n
REP STOS m8	F3 AA	Fill CX bytes at ES:[DI] with AL	6+9n	5/ 7+4D	4+ (D+3)n
REP STOS m16	F3 AB	Fill CX words at ES:[DI] with AX	6+9n	5/ 7+4D	4+ (D+3)n
REPE CMPS m8,m8	F3 A6	Find nonmatching bytes in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPE CMPS m16,m16	F3 A7	Find nonmatching words in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPE SCAS m8	F3 AE	Find non-AL byte starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPE SCAS m16	F3 AF	Find non-AX word starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPNE CMPS m8,m8	F2 A6	Find matching bytes in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPNE CMPS	F2 A7	Find matching words in ES:[DI] and	5+22n	5/	4+

Mnemonic	Opcode	Description	186	486	DSTni
m16,m16		segment:[SI]		7+7D	(D+3)n
REPNE SCAS m8	F2 AE	Find AL byte starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPNE SCAS m16	F2 AF	Find AX word starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPNZ CMPS m8,m8	F2 A6	Find matching bytes in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPNZ CMPS m16,m16	F2 A7	Find matching words in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPNZ SCAS m8	F2 AE	Find AL byte starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPNZ SCAS m16	F2 AF	Find AX word starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPZ CMPS m8,m8	F3 A6	Find nonmatching bytes in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPZ CMPS m16,m16	F3 A7	Find nonmatching words in ES:[DI] and segment:[SI]	5+22n	5/ 7+7D	4+ (D+3)n
REPZ SCAS m8	F3 AE	Find non-AL byte starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
REPZ SCAS m16	F3 AF	Find non-AX word starting at ES:[DI]	5+15n	5/ 7+5D	4+ (D+3)n
RET	C3	Return near to calling procedure	16	5	3
RET	CB	Return far to calling procedure	22	13	5
RET imm16	C2 iw	Return near; pop imm16 parameters	18	5	7
RET imm16	CA iw	Return far; pop imm16	25	14	9
ROL r/m8,1	D0 /0	Rotate 8 bits of r/m byte left once	2/15	3 / 4	1
ROL r/m8,CL	D2 /0	Rotate 8 bits of r/m byte left CL times	5+n/ 17+n	3 / 4	4+n
ROL r/m8,imm8	C0 /0 ib	Rotate 8 bits of r/m byte left imm8 times	5+n/ 17+n	2 / 4	4+n
ROL r/m16,1	D1 /0	Rotate 8 bits of r/m word left once	2/15	3 / 4	1
ROL r/m16,CL	D3 /0	Rotate 8 bits of r/m word left CL times	5+n/ 17+n	3 / 4	4+n
ROL r/m16,imm8	C1 /0 ib	Rotate 8 bits of r/m word left imm8 times	5+n/ 17+n	2 / 4	4+n
ROR r/m8,1	D0 /1	Rotate 8 bits of r/m byte right once	2/15	3 / 4	1
ROR r/m8,CL	D2 /1	Rotate 8 bits of r/m byte right CL times	5+n/ 17+n	3 / 4	4+n
ROR r/m8,imm8	C0 /1 ib	Rotate 8 bits of r/m byte right imm8 times	5+n/ 17+n	2 / 4	4+n
ROR r/m16,1	D1 /1	Rotate 8 bits of r/m word right once	2/15	3 / 4	1
ROR r/m16,CL	D3 /1	Rotate 8 bits of r/m word right CL times	5+n/ 17+n	3 / 4	4+n
ROR r/m16,imm8	C1 /1 ib	Rotate 8 bits of r/m word right imm8 times	5+n/ 17+n	2 / 4	4+n
SAHF	9E	Store AH in low byte of the Processor Status Flags register	3	2	1
SAL r/m8,1	D0 /4	Multiply r/m byte by 2, once	2/15	3 / 4	1
SAL r/m8,CL	D2 /4	Multiply r/m byte by 2, CL times	5+n/ 17+n	3 / 4	4+n
SAL	C0 /4 ib	Multiply r/m byte by 2, imm8 times	5+n/	2 / 4	4+n

Mnemonic	Opcode	Description	186	486	DSTni
r/m8,imm8			17+n		
SAL r/m16,1	D1 /4	Multiply r/m word by 2, once	2/15	3 / 4	1
SAL r/m16,CL	D3 /4	Multiply r/m word by 2, CL times	5+n/ 17+n	3 / 4	4+n
SAL r/m16,imm8	C1 /4 ib	Multiply r/m word by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SAR r/m8,1	D0 /7	Perform a signed division of r/m byte by 2, once	2/15	3 / 4	1
SAR r/m8,CL	D2 /7	Perform a signed division of r/m byte by 2, CL times	5+n/ 17+n	3 / 4	4+n
SAR r/m8,imm8	C0 /7 ib	Perform a signed division of r/m byte by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SAR r/m16,1	D1 /7	Perform a signed division of r/m word by 2, once	2/15	3 / 4	1
SAR r/m16,CL	D3 /7	Perform a signed division of r/m word by 2, CL times	5+n/ 17+n	3 / 4	4+n
SAR r/m16,imm8	C1 /7 ib	Perform a signed division of r/m word by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SBB AL,imm8	1C ib	Subtract immediate byte from AL with borrow	3	1	1
SBB AX,imm16	1D iw	Subtract immediate word from AX with borrow	4	1	1
SBB r/m8,imm8	80 /3 ib	Subtract immediate byte from r/m byte with borrow	4/16	1/3	1/4
SBB r/m16,imm16	81 /3 iw	Subtract immediate word from r/m word with borrow	4/16	1/3	1/4
SBB r/m16,imm8	83 /3 ib	Subtract sign-extended immediate byte from r/m word with borrow	4/16	1/3	1/4
SBB r/m8,r8	18 /r	Subtract byte register from r/m byte with borrow	3/10	1/3	1/4
SBB r/m16,r16	19 /r	Subtract word register from r/m word with borrow	3/10	1/3	1/4
SBB r8,r/m8	1A /r	Subtract r/m byte from byte register with borrow	3/10	1/3	1/4
SBB r16,r/m16	1B /r	Subtract r/m word from word register with borrow	3/10	1/3	1/4
SCAS m8	AE	Compare byte AL to ES:[DI]; Update DI	15	6	4
SCAS m16	AF	Compare word AX to ES:[DI]; Update DI	15	6	4
SCASB	AE	Compare byte AL to ES:[DI]; Update DI	15	6	4
SCASW	AF	Compare word AX to ES:[DI]; Update DI	15	6	4
SHL r/m8,1	D0 /4	Multiply r/m byte by 2, once	2/15	3 / 4	1
SHL r/m8,CL	D2 /4	Multiply r/m byte by 2, CL times	5+n/ 17+n	3 / 4	4+n
SHL r/m8,imm8	C0 /4 ib	Multiply r/m byte by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SHL r/m16,1	D1 /4	Multiply r/m word by 2, once	2/15	3 / 4	1
SHL r/m16,CL	D3 /4	Multiply r/m word by 2, CL times	5+n/ 17+n	3 / 4	4+n
SHL r/m16,imm8	C1 /4 ib	Multiply r/m word by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SHR r/m8,1	D0 /5	Divide unsigned r/m byte by 2, once	2/15	3 / 4	1

Mnemonic	Opcode	Description	186	486	DSTni
SHR r/m8,CL	D2 /5	Divide unsigned r/m byte by 2, CL times	5+n/ 17+n	3 / 4	4+n
SHR r/m8,imm8	C0 /5 ib	Divide unsigned r/m byte by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
SHR r/m16,1	D1 /5	Divide unsigned r/m word by 2, once	2/15	3 / 4	1
SHR r/m16,CL	D3 /5	Divide unsigned r/m word by 2, CL times	5+n/ 17+n	3 / 4	4+n
SHR r/m16,imm8	C1 /5 ib	Divide unsigned r/m word by 2, imm8 times	5+n/ 17+n	2 / 4	4+n
STC	F9	Set the Carry Flag to 1	2	2	1
STD	FD	Set the Direction Flag so the Source Index (SI) and/or the destination Index (DI) registers will decrement during string instructions	2	2	1
STI	FB	Enable maskable interrupts after the next instruction	2	5	1
STOS m8	AA	Store AL in byte ES:[DI]; Update DI	10	5	2
STOS m16	AB	Store AX in word ES:[DI]; Update DI	10	5	2
STOSB	AA	Store AL in byte ES:[DI]; Update DI	10	5	2
STOSW	AB	Store AX in word ES:[DI]; Update DI	10	5	2
SUB AL,imm8	2C ib	Subtract immediate byte from AL	3	1	1
SUB AX,imm16	2D iw	Subtract immediate word from AX	4	1	1
SUB r/m8,imm8	80 /5 ib	Subtract immediate byte from r/m byte	4/16	1/3	1/4
SUB r/m16,imm16	81 /5 iw	Subtract immediate word from r/m word	4/16	1/3	1/4
SUB r/m16,imm8	83 /5 ib	Subtract sign-extended immediate byte from r/m word	4/16	1/3	1/4
SUB r/m8,r8	28 /r	Subtract byte register from r/m byte	3/10	1/3	1/4
SUB r/m8,r16	29 /r	Subtract word register from r/m word	3/10	1/3	1/4
SUB r8,r/m8	2A /r	Subtract r/m byte from byte register	4/16	1/3	1/4
SUB r16,r/m16	2B /r	Subtract r/m word from word register	4/16	1/3	1/4
TEST AL,imm8	A8 ib	AND immediate byte with AL	3	1	1
TEST AX,imm16	A9 iw	AND immediate word with AX	4	1	1
TEST r/m8,imm8	F6 /0 ib	AND immediate byte with r/m byte	4/10	1 / 2	1/4
TEST r/m16,imm16	F7 /0 iw	AND immediate word with r/m word	4/10	1 / 2	1/4
TEST r/m8,r8	84 /r	AND byte register with r/m byte	3/10	1 / 2	1/4
TEST r/m16,r16	85 /r	AND word register with r/m word	3/10	1 / 2	1/4
WAIT	9B	Performs a NOP	?	?	?
XCHG AX,r16	90+rw	Exchange word register with AX	3	3	3
XCHG r16,AX	90+rw	Exchange AX with word register	3	3	3

Mnemonic	Opcode	Description	186	486	DSTni
XCHG r/m8,r8	86 /r	Exchange byte register with r/m byte	4/17	3/5	3/7
XCHG r8,r/m8	86 /r	Exchange r/m byte with byte register	4/17	3/5	3/7
XCHG r/m16,r16	87 /r	Exchange word register with r/m word	4/17	3/5	3/7
XCHG r16,r/m16	87 /r	Exchange r/m word with word register	4/17	3/5	3/7
XLAT m8	D7	Set AL to memory byte segment:[BX+unsigned AL]	11	4	5
XLATB	D7	Set AL to memory byte segment:[BX+unsigned AL]	11	4	5
XOR AL,imm8	34 ib	XOR immediate byte with AL	3	1	1
XOR AX,imm16	35 iw	XOR immediate word with AX	4	1	1
XOR r/m8,imm8	80 /6 ib	XOR immediate byte with r/m byte	4/16	1/3	1/4
XOR r/m16,imm16	81 /6 iw	XOR immediate word with r/m word	4/16	1/3	1/4
XOR r/m16,imm8	83 /6 ib	XOR sign-extended immediate word with r/m word	4/16	1/3	1/4
XOR r/m8,r8	30 /r	XOR byte register with r/m byte	3/10	1/3	1/4
XOR r/m16,r16	31 /r	XOR word register with r/m word	3/10	1/3	1/4
XOR r8,r/m8	32 /r	XOR r/m byte with byte register	3/10	1/3	1/4
XOR r16,r/m16	33 /r	XOR r/m word with word register	3/10	1/3	1/4

Each instruction includes an instruction mnemonic and zero or more operands. A placeholder is shown for operands that must be provided. The placeholder indicates the size and type of operand that is allowed.

The Operand	Is a placeholder for
imm8	An immediate byte: a signed number between –128 and 127
imm16	An immediate word: a signed number between –32768 and 32767
m	An operand in memory
m8	A byte string in memory pointed to by DS:SI or ES:DI
m16	A word string in memory pointed to by DS:SI or ES:DI
m16&16	A pair of words in memory
m16:16	A doubleword in memory that contains a signed, relative offset displacement
moffs8	A byte in memory that contains a signed, relative offset displacement
ptr16:16	A full address (segment:offset)
r8	A general byte register: AL, BL, CL, DL, AH, BH, CH or DH
r16	A general word register: AX, BX, CX, DX, BP, SP, DI or SI
r/m8	A general byte register or a byte in memory

r/m16	A general word register or a word in memory
rel8	A signed, relative offset displacement between –128 and 127
rel16	A signed, relative offset displacement between –32768 and 32767
sreg	A segment register
Parameter	Indicates that
/0-/7	The Auxiliary field in the Operand Address byte specifies an extension from 0 to 7 to the opcode instead of a register.
/r	The Auxiliary field in the Operand Address byte specifies a register instead of an opcode extension. If the Opcode byte specifies a byte register, the registers are assigned as follows: AL=0, CL=1, DL=2, BL=3, AH=4, CH=5, DH=6 and BH=7. If the Opcode byte specifies a word register, the registers are assigned as follows: AX=0, CX=1, DX=2, BX=3, SP=4, BP=5, SI=6 and DI=7.
/sr	The Auxiliary field in the Operand Address byte specifies a segment register as follows: ES=0, CS=1, SS=2 and DS=3.
cb	The byte following the Opcode byte specifies an offset.
cd	The doubleword following the Opcode byte specifies an offset and in some cases a segment.
cw	The word following the Opcode byte specifies an offset and in some cases a segment
ib	The parameter is an immediate byte. The Opcode byte determines whether it is interpreted as a signed or unsigned number.
iw	The parameter is an immediate word. The Opcode byte determines whether it is interpreted as a signed or unsigned number.
rb	The byte register operand is specified in the Opcode byte. To determine the Opcode byte for a particular register, add the hexadecimal value on the left of the plus sign to the value of rb for that register as follows: AL=0, CL=1, DL=2, BL=3, AH=4, CH=5, DH=6 and BH=7
rw	The word register operand is specified in the Opcode byte. To determine the Opcode byte for a particular register, add the hexadecimal value on the left of the plus sign to the value of rw for that register as follows: AX=0, CX=1, DX=2, BX=3, SP=4, BP=5, SI=6 and DI=7.
/	This number of clocks required for a register operand is different the number required for an operand located in memory. The number to the left corresponds with a register operand. The number to the right corresponds with an operand located in memory.
,	The number of clocks depends on the result of the condition tested. The number to the left corresponds with a True or Pass result, and the number to the right corresponds with a False or Fail result.
n	The number of clocks depends on the number of times the instruction is repeated. n is the number of repetitions.
I	Interrupt handler +jump=18
J	Jump / queue init=5
D	Duration of instruction repeated

8: DSTni Sample Code

```
DstExStd.h  This header file defines the "standard" constant
               values used for all DSTni-EX module code development.

10-DEC-2002    WD          Author, creation. Revision 1.00

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#if !defined(DSTEXSTD_H)
#define DSTEXSTD_H

#include "DstExConfig.h"

#if !defined(DST_REG_PCB)
       #define DST_REG_PCB           0xFF00
#endif

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Peripheral Control Block register addresses
** Base address defined in DstLxConfig.h
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
/* System control and configuration registers*/
#define DST_REG_RELREG (DST_REG_PCB + 0x00FE) /* Relocation Register     */
#define DST_REG_DCR           (DST_REG_PCB + 0x00F8) /* DSTni Configuration Reg */
#define DST_REG_RCR           (DST_REG_PCB + 0x00F6) /* Reset Configuration Reg */
#define DST_REG_PRL           (DST_REG_PCB + 0x00F4) /* Processor Release Reg   */
#define DST_REG_AUXCON (DST_REG_PCB + 0x00F2) /* Auxiliary Configuration  */
#define DST_REG_SYSCON        (DST_REG_PCB + 0x00F0) /* System Configuration    */

/* Chip select registers */
#define DST_REG_CAR           (DST_REG_PCB + 0x00AE) /* Checksum Adder           */
#define DST_REG_CDR           (DST_REG_PCB + 0x00AC) /* Checksum Data            */
#define DST_REG_PRCS          (DST_REG_PCB + 0x00AA) /* Page Chip Select         */
#define DST_REG_MPCS          (DST_REG_PCB + 0x00A8) /* Memory / Peripheral Ctrl */
#define DST_REG_MMCS          (DST_REG_PCB + 0x00A6) /* Mid Memory Chip Select   */
#define DST_REG_PACS          (DST_REG_PCB + 0x00A4) /* Peripheral Chip Select   */
#define DST_REG_LMCS          (DST_REG_PCB + 0x00A2) /* Lower Memory Chip Select*/
#define DST_REG_UMCS   (DST_REG_PCB + 0x00A0) /* Upper Memory Chip Select */
```

```
/* Phase lock loop, CPU clock and SPI control */
#define DST_REG_PLLCLKUSB     (DST_REG_PCB + 0x006E)

/* Watchdog timer register */
#define DST_REG_WDTCON        (DST_REG_PCB + 0x006C)

/* Random number generator / Linear counter */
#define DST_REG_RNG           (DST_REG_PCB + 0x006A)

/* SDRAM control */
#define DST_REG_SDRAMCTRL     (DST_REG_PCB + 0x0068) /* Control register */
#define DST_REG_REFMAX        (DST_REG_PCB + 0x0064) /* Refresh maxcount */

/* Timer 2 registers */
#define DST_REG_T2CON         (DST_REG_PCB + 0x0066) /* Control             */
#define DST_REG_T2CMPA        (DST_REG_PCB + 0x0062) /* Compare A           */
                                                   /* No Compare B for T2 */
#define DST_REG_T2CNT         (DST_REG_PCB + 0x0060) /* Count               */

/* Timer 1 Registers */
#define DST_REG_T1CON         (DST_REG_PCB + 0x005E) /* Control   */
#define DST_REG_T1CMPB        (DST_REG_PCB + 0x005C) /* Compare B */
#define DST_REG_T1CMPA        (DST_REG_PCB + 0x005A) /* Compare A */
#define DST_REG_T1CNT         (DST_REG_PCB + 0x0058) /* Count     */

/* Timer 0 Registers */
#define DST_REG_T0CON         (DST_REG_PCB + 0x0056) /* Control   */
#define DST_REG_T0CMPB        (DST_REG_PCB + 0x0054) /* Compare B */
#define DST_REG_T0CMPA        (DST_REG_PCB + 0x0052) /* Compare A */
#define DST_REG_T0CNT         (DST_REG_PCB + 0x0050) /* Count     */

/* Interrupt controller Master Mode registers */
#define DST_REG_CANCON        (DST_REG_PCB + 0x004C) /* CAN 1 & I6 control   */
#define DST_REG_DMA3CON       (DST_REG_PCB + 0x004A) /* DMA 3 interrupt control */
#define DST_REG_DMA2CON       (DST_REG_PCB + 0x0048) /* DMA 2 interrupt control */
#define DST_REG_SP3CON        (DST_REG_PCB + 0x0046) /* SP3 interrupt control   */
#define DST_REG_SP0CON        (DST_REG_PCB + 0x0044) /* Serial Port 0 int ctrl  */
#define DST_REG_SP1CON        (DST_REG_PCB + 0x0042) /* Serial Port 1 int ctrl  */
#define DST_REG_SP2CON        (DST_REG_PCB + 0x0040) /* Serial Port 2 int ctrl  */
#define DST_REG_I3CON         (DST_REG_PCB + 0x003E) /* INT3 interrupt control  */
#define DST_REG_USBCON        (DST_REG_PCB + 0x003E) /* USB interrupt control   */
#define DST_REG_SPICON        (DST_REG_PCB + 0x003C) /* SPI interrupt control   */
#define DST_REG_I2CCON        (DST_REG_PCB + 0x003C) /* I2C interrupt control   */
#define DST_REG_I1CON         (DST_REG_PCB + 0x003A) /* INT1 interrupt control  */
#define DST_REG_MAC1CON       (DST_REG_PCB + 0x003A) /* MAC1 interrupt control  */
#define DST_REG_MAC0CON       (DST_REG_PCB + 0x0038) /* MAC0 interrupt control  */
#define DST_REG_DMA1CON       (DST_REG_PCB + 0x0036) /* DMA 1 interrupt control */
#define DST_REG_DMA0CON       (DST_REG_PCB + 0x0034) /* DMA 0 interrupt control */
#define DST_REG_TCUCON        (DST_REG_PCB + 0x0032) /* Timer/Counter Unit      */
#define DST_REG_INTSTS        (DST_REG_PCB + 0x0030) /* Interrupt Status        */
#define DST_REG_REQST         (DST_REG_PCB + 0x002E) /* Interrupt Request       */
#define DST_REG_INSERV        (DST_REG_PCB + 0x002C) /* In-Service              */
#define DST_REG_PRIMSK        (DST_REG_PCB + 0x002A) /* Priority mask           */
#define DST_REG_IMASK         (DST_REG_PCB + 0x0028) /* Interrupt mask          */
#define DST_REG_POLLST        (DST_REG_PCB + 0x0026) /* Interrupt poll status   */
#define DST_REG_POLL          (DST_REG_PCB + 0x0024) /* Interrupt poll          */
#define DST_REG_EOI           (DST_REG_PCB + 0x0022) /* End Of Interrupt        */

/* DMA 0 control registers */
#define DST_REG_D0CON         (DST_REG_PCB + 0x00CA) /* Control */
#define DST_REG_D0TC          (DST_REG_PCB + 0x00C8) /* Terminal Count */
#define DST_REG_D0DSTH        (DST_REG_PCB + 0x00C6) /* Destination hi byte */
#define DST_REG_D0DSTL        (DST_REG_PCB + 0x00C4) /* Destination lo byte */
#define DST_REG_D0SRCH        (DST_REG_PCB + 0x00C2) /* Source hi byte */
#define DST_REG_D0SRCL        (DST_REG_PCB + 0x00C0) /* Source lo byte */
```

```
/* DMA 1 control registers */
#define DST_REG_D1CON          (DST_REG_PCB + 0x00DA) /* Control */
#define DST_REG_D1TC           (DST_REG_PCB + 0x00D8) /* Terminal Count */
#define DST_REG_D1DSTH         (DST_REG_PCB + 0x00D6) /* Destination hi byte */
#define DST_REG_D1DSTL         (DST_REG_PCB + 0x00D4) /* Destination lo byte */
#define DST_REG_D1SRCH         (DST_REG_PCB + 0x00D2) /* Source hi byte */
#define DST_REG_D1SRCL         (DST_REG_PCB + 0x00D0) /* Source lo byte */

/* DMA 2 control registers */
#define DST_REG_D2CON          (DST_REG_PCB + 0x009A)   /* Control */
#define DST_REG_D2TC           (DST_REG_PCB + 0x0098)   /* Terminal Count */
#define DST_REG_D2DSTH         (DST_REG_PCB + 0x0096)   /* Destination hi byte */
#define DST_REG_D2DSTL         (DST_REG_PCB + 0x0094)   /* Destination lo byte */
#define DST_REG_D2SRCH         (DST_REG_PCB + 0x0092)   /* Source hi byte */
#define DST_REG_D2SRCL         (DST_REG_PCB + 0x0090)   /* Source lo byte */

/* DMA 3 control registers */
#define DST_REG_D3CON          (DST_REG_PCB + 0x00BA)   /* Control */
#define DST_REG_D3TC           (DST_REG_PCB + 0x00B8)   /* Terminal Count */
#define DST_REG_D3DSTH         (DST_REG_PCB + 0x00B6)   /* Destination hi byte */
#define DST_REG_D3DSTL         (DST_REG_PCB + 0x00B4)   /* Destination lo byte */
#define DST_REG_D3SRCH         (DST_REG_PCB + 0x00B2)   /* Source hi byte */
#define DST_REG_D3SRCL         (DST_REG_PCB + 0x00B0)   /* Source lo byte */

/* Serial Port 3 registers */
#define DST_REG_SP3AUX         (DST_REG_PCB + 0x00EA)   /* Auxiliary control */
#define DST_REG_SP3BAUD        (DST_REG_PCB + 0x00E8)   /* Baud rate divisor */
#define DST_REG_SP3RD          (DST_REG_PCB + 0x00E6)   /* Receive Data */
#define DST_REG_SP3TD          (DST_REG_PCB + 0x00E4)   /* Transmit Data */
#define DST_REG_SP3STS         (DST_REG_PCB + 0x00E2)   /* Status */
#define DST_REG_SP3CT          (DST_REG_PCB + 0x00E0)   /* Control */

/* Serial Port 0 registers */
#define DST_REG_SP0AUX         (DST_REG_PCB + 0x008A)   /* Auxiliary control */
#define DST_REG_SP0BAUD        (DST_REG_PCB + 0x0088)   /* Baud rate divisor */
#define DST_REG_SP0RD          (DST_REG_PCB + 0x0086)   /* Receive Data */
#define DST_REG_SP0TD          (DST_REG_PCB + 0x0084)   /* Transmit Data */
#define DST_REG_SP0STS         (DST_REG_PCB + 0x0082)   /* Status */
#define DST_REG_SP0CT          (DST_REG_PCB + 0x0080)   /* Control */

/* Serial Port 1 registers */
#define DST_REG_SP1AUX         (DST_REG_PCB + 0x001A)   /* RS422/RS485 Control */
#define DST_REG_SP1BAUD        (DST_REG_PCB + 0x0018)   /* Baud rate divisor */
#define DST_REG_SP1RD          (DST_REG_PCB + 0x0016)   /* Receive Data */
#define DST_REG_SP1TD          (DST_REG_PCB + 0x0014)   /* Transmit Data */
#define DST_REG_SP1STS         (DST_REG_PCB + 0x0012)   /* Status */
#define DST_REG_SP1CT          (DST_REG_PCB + 0x0010)   /* Control */

/* Serial Port 2 registers */
#define DST_REG_SP2AUX         (DST_REG_PCB + 0x000A)   /* Auxiliary control */
#define DST_REG_SP2BAUD        (DST_REG_PCB + 0x0008)   /* Baud rate divisor */
#define DST_REG_SP2RD          (DST_REG_PCB + 0x0006)   /* Receive Data */
#define DST_REG_SP2TD          (DST_REG_PCB + 0x0004)   /* Transmit Data */
#define DST_REG_SP2STS         (DST_REG_PCB + 0x0002)   /* Status */
#define DST_REG_SP2CT          (DST_REG_PCB + 0x0000)   /* Control */

/* LED Control Register */
#define DST_REG_LEDC           (DST_REG_PCB + 0x007E)

/* Programmable I/O Bank 1 (bits 31-16) registers */
#define DST_REG_PIODATA1       (DST_REG_PCB + 0x007A)   /* Input / Output Data */
#define DST_REG_PIODIR1        (DST_REG_PCB + 0x0078)   /* Direction Select */
#define DST_REG_PIOMODE1       (DST_REG_PCB + 0x0076)   /* Mode Select */

/* Programmable I/O Bank 0 (bits 15-0) registers */
#define DST_REG_PIODATA0       (DST_REG_PCB + 0x0074)   /* Input / Output Data */
#define DST_REG_PIODIR0        (DST_REG_PCB + 0x0072)   /* Direction Select */
#define DST_REG_PIOMODE0       (DST_REG_PCB + 0x0070)   /* Mode Select */
```

```
/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Peripheral device I/O addresses
** These are the peripheral devices not part of the Peripheral Control Block
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Ethernet Controller Registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_ETH0_BASE         0x9000
#define DST_ETH1_BASE         0x9100

#define DST_REG_ETH0_RDP      (DST_ETH0_BASE + 0x0010)  /* Register Data Port */
#define DST_REG_ETH0_RAP      (DST_ETH0_BASE + 0x0012)  /* Register Access Port */
#define DST_REG_ETH0_RST      (DST_ETH0_BASE + 0x0014)  /* Reset Port */
#define DST_REG_ETH0_MII      (DST_ETH0_BASE + 0x0018)  /* MII Port */

#define DST_REG_ETH1_RDP      (DST_ETH1_BASE + 0x0010)  /* Register Data Port */
#define DST_REG_ETH1_RAP      (DST_ETH1_BASE + 0x0012)  /* Register Access Port */
#define DST_REG_ETH1_RST      (DST_ETH1_BASE + 0x0014)  /* Reset Port */
#define DST_REG_ETH1_MII      (DST_ETH1_BASE + 0x0018)  /* MII Port */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** CAN 0 Registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_CAN0_BASE             0xA800
#define DST_CAN1_BASE             0xA900

/* Transmit message 0 */
#define DST_REG_CAN0_TX0_ID28_13  (DST_CAN0_BASE + 0x00)  /* Hi ID bits */
#define DST_REG_CAN0_TX0_ID12_00  (DST_CAN0_BASE + 0x02)  /* Lo ID bits */
#define DST_REG_CAN0_TX0_D63_48      (DST_CAN0_BASE + 0x04)  /* Hi Data bits */
#define DST_REG_CAN0_TX0_D47_32      (DST_CAN0_BASE + 0x06)  /* .... */
#define DST_REG_CAN0_TX0_D31_16      (DST_CAN0_BASE + 0x08)  /* .... */
#define DST_REG_CAN0_TX0_D15_00      (DST_CAN0_BASE + 0x0A)  /* Lo Data bits */
#define DST_REG_CAN0_TX0_LEN         (DST_CAN0_BASE + 0x0C)  /* Length */
#define DST_REG_CAN0_TX0_CTRL        (DST_CAN0_BASE + 0x0E)  /* Control */

/* Transmit message 1 */
#define DST_REG_CAN0_TX1_ID28_13     (DST_CAN0_BASE + 0x10)  /* Hi ID bits */
#define DST_REG_CAN0_TX1_ID12_00     (DST_CAN0_BASE + 0x12)  /* Lo ID bits */
#define DST_REG_CAN0_TX1_D63_48      (DST_CAN0_BASE + 0x14)  /* Hi Data bits */
#define DST_REG_CAN0_TX1_D47_32      (DST_CAN0_BASE + 0x16)  /* .... */
#define DST_REG_CAN0_TX1_D31_16      (DST_CAN0_BASE + 0x18)  /* .... */
#define DST_REG_CAN0_TX1_D15_00      (DST_CAN0_BASE + 0x1A)  /* Lo Data bits */
#define DST_REG_CAN0_TX1_LEN         (DST_CAN0_BASE + 0x1C)  /* Length */
#define DST_REG_CAN0_TX1_CTRL        (DST_CAN0_BASE + 0x1E)  /* Control */

/* Transmit message 2 */
#define DST_REG_CAN0_TX2_ID28_13     (DST_CAN0_BASE + 0x20)  /* Hi ID bits */
#define DST_REG_CAN0_TX2_ID12_00     (DST_CAN0_BASE + 0x22)  /* Lo ID bits */
#define DST_REG_CAN0_TX2_D63_48      (DST_CAN0_BASE + 0x24)  /* Hi Data bits */
#define DST_REG_CAN0_TX2_D47_32      (DST_CAN0_BASE + 0x26)  /* .... */
#define DST_REG_CAN0_TX2_D31_16      (DST_CAN0_BASE + 0x28)  /* .... */
#define DST_REG_CAN0_TX2_D15_00      (DST_CAN0_BASE + 0x2A)  /* Lo Data bits */
#define DST_REG_CAN0_TX2_LEN         (DST_CAN0_BASE + 0x2C)  /* Length */
#define DST_REG_CAN0_TX2_CTRL        (DST_CAN0_BASE + 0x2E)  /* Control */
```

```
/* Receive */
#define DST_REG_CAN0_RX_ID28_13       (DST_CAN0_BASE + 0x30)  /* Hi ID bit */
#define DST_REG_CAN0_RX_ID12_00       (DST_CAN0_BASE + 0x32)  /* Lo ID bits */
#define DST_REG_CAN0_RX_D63_48        (DST_CAN0_BASE + 0x34)  /* Hi Data bits */
#define DST_REG_CAN0_RX_D47_32        (DST_CAN0_BASE + 0x36)  /* .... */
#define DST_REG_CAN0_RX_D31_16        (DST_CAN0_BASE + 0x38)  /* .... */
#define DST_REG_CAN0_RX_D15_00        (DST_CAN0_BASE + 0x3A)  /* Lo Data bits */
#define DST_REG_CAN0_RX_LEN           (DST_CAN0_BASE + 0x3C)  /* Length */
#define DST_REG_CAN0_RX_FLAGS         (DST_CAN0_BASE + 0x3E)  /* Control */

/* Status/IRQ registers */
#define DST_REG_CAN0_ERR_CNT          (DST_CAN0_BASE + 0x40)  /* Rx/Tx err count  */
#define DST_REG_CAN0_ERR_STAT_CNT     (DST_CAN0_BASE + 0x42)  /* Err Status count */
#define DST_REG_CAN0_MSG_LEVEL        (DST_CAN0_BASE + 0x44)  /* Msg Level Thresh */
#define DST_REG_CAN0_IRQ_FLAGS        (DST_CAN0_BASE + 0x46)  /* Interrupt Flags */
#define DST_REG_CAN0_IRQ_ENABLE       (DST_CAN0_BASE + 0x48)  /* Interrupt enable */

/* CAN Configuration */
#define DST_REG_CAN0_MODE             (DST_CAN0_BASE + 0x4A)  /* Operating mode */
#define DST_REG_CAN0_BITRATE          (DST_CAN0_BASE + 0x4C)  /* */
#define DST_REG_CAN0_TIMING           (DST_CAN0_BASE + 0x4E)  /* */

/* Acceptance Mask and Code registers */
#define DST_REG_CAN0_FILTER_ENA       (DST_CAN0_BASE + 0x50)  /* Filter enables */

#define DST_REG_CAN0_AMR0_ID28_13     (DST_CAN0_BASE + 0x52) /*Acceptance mask 0*/
#define DST_REG_CAN0_AMR0_ID12_00     (DST_CAN0_BASE + 0x54)  /* .. */
#define DST_REG_CAN0_AMR0_D63_48      (DST_CAN0_BASE + 0x56)  /* .. */
#define DST_REG_CAN0_ACR0_ID28_13     (DST_CAN0_BASE + 0x58)  /* Acceptance code 0*/
#define DST_REG_CAN0_ACR0_ID12_00     (DST_CAN0_BASE + 0x5A)  /* .. */
#define DST_REG_CAN0_ACR0_D63_48      (DST_CAN0_BASE + 0x5C)  // ..

#define DST_REG_CAN0_AMR1_ID28_13     (DST_CAN0_BASE + 0x5E) /*Acceptance mask 1*/
#define DST_REG_CAN0_AMR1_ID12_00     (DST_CAN0_BASE + 0x60)  /* .. */
#define DST_REG_CAN0_AMR1_D63_48      (DST_CAN0_BASE + 0x62)  /* .. */
#define DST_REG_CAN0_ACR1_ID28_13     (DST_CAN0_BASE + 0x64)  /* Acceptance code 1*/
#define DST_REG_CAN0_ACR1_ID12_00     (DST_CAN0_BASE + 0x66)  /* .. */
#define DST_REG_CAN0_ACR1_D63_48      (DST_CAN0_BASE + 0x68)  /* .. */

#define DST_REG_CAN0_AMR2_ID28_13     (DST_CAN0_BASE + 0x6A)  /*Acceptance mask 2*/
#define DST_REG_CAN0_AMR2_ID12_00     (DST_CAN0_BASE + 0x6C)  /* .. */
#define DST_REG_CAN0_AMR2_D63_48      (DST_CAN0_BASE + 0x6E)  /* .. */
#define DST_REG_CAN0_ACR2_ID28_13     (DST_CAN0_BASE + 0x70)  /* Acceptance code 2*/
#define DST_REG_CAN0_ACR2_ID12_00     (DST_CAN0_BASE + 0x72)  /* .. */
#define DST_REG_CAN0_ACR2_D63_48      (DST_CAN0_BASE + 0x74)  /* .. */

/* Analysis registers */
#define DST_REG_CAN0_ALCR             (DST_CAN0_BASE + 0x76)  /* Arb. Lost Cap.Reg*/
#define DST_REG_CAN0_ECR              (DST_CAN0_BASE + 0x78)  /* Error Capture Reg*/
#define DST_REG_CAN0_FRR              (DST_CAN0_BASE + 0x7A)  /* Frame Refer Reg  */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** CAN 1 Registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/* Transmit message 0 */
#define DST_REG_CAN1_TX0_ID28_13      (DST_CAN1_BASE + 0x00)  /* Hi ID bits */
#define DST_REG_CAN1_TX0_ID12_00      (DST_CAN1_BASE + 0x02)  /* Lo ID bits */
#define DST_REG_CAN1_TX0_D63_48       (DST_CAN1_BASE + 0x04)  /* Hi Data bits */
#define DST_REG_CAN1_TX0_D47_32       (DST_CAN1_BASE + 0x06)  /* .... */
#define DST_REG_CAN1_TX0_D31_16       (DST_CAN1_BASE + 0x08)  /* .... */
#define DST_REG_CAN1_TX0_D15_00       (DST_CAN1_BASE + 0x0A)  /* Lo Data bits */
#define DST_REG_CAN1_TX0_LEN          (DST_CAN1_BASE + 0x0C)  /* Length */
#define DST_REG_CAN1_TX0_CTRL         (DST_CAN1_BASE + 0x0E)  /* Control */
```

```
/* Transmit message 1 */
#define DST_REG_CAN1_TX1_ID28_13      (DST_CAN1_BASE + 0x10)  /* Hi ID bits */
#define DST_REG_CAN1_TX1_ID12_00      (DST_CAN1_BASE + 0x12)  /* Lo ID bits */
#define DST_REG_CAN1_TX1_D63_48       (DST_CAN1_BASE + 0x14)  /* Hi Data bits */
#define DST_REG_CAN1_TX1_D47_32       (DST_CAN1_BASE + 0x16)  /* .... */
#define DST_REG_CAN1_TX1_D31_16       (DST_CAN1_BASE + 0x18)  /* .... */
#define DST_REG_CAN1_TX1_D15_00       (DST_CAN1_BASE + 0x1A)  /* Lo Data bits */
#define DST_REG_CAN1_TX1_LEN          (DST_CAN1_BASE + 0x1C)  /* Length */
#define DST_REG_CAN1_TX1_CTRL         (DST_CAN1_BASE + 0x1E)  /* Control */

/* Transmit message 2 */
#define DST_REG_CAN1_TX2_ID28_13      (DST_CAN1_BASE + 0x20)  /* Hi ID bits */
#define DST_REG_CAN1_TX2_ID12_00      (DST_CAN1_BASE + 0x22)  /* Lo ID bits */
#define DST_REG_CAN1_TX2_D63_48       (DST_CAN1_BASE + 0x24)  /* Hi Data bits */
#define DST_REG_CAN1_TX2_D47_32       (DST_CAN1_BASE + 0x26)  /* .... */
#define DST_REG_CAN1_TX2_D31_16       (DST_CAN1_BASE + 0x28)  /* .... */
#define DST_REG_CAN1_TX2_D15_00       (DST_CAN1_BASE + 0x2A)  /* Lo Data bits */
#define DST_REG_CAN1_TX2_LEN          (DST_CAN1_BASE + 0x2C)  /* Length */
#define DST_REG_CAN1_TX2_CTRL         (DST_CAN1_BASE + 0x2E)  /* Control */

/* Receive */
#define DST_REG_CAN1_RX_ID28_13       (DST_CAN1_BASE + 0x30)  /* Hi ID bit */
#define DST_REG_CAN1_RX_ID12_00       (DST_CAN1_BASE + 0x32)  /* Lo ID bits */
#define DST_REG_CAN1_RX_D63_48        (DST_CAN1_BASE + 0x34)  /* Hi Data bits */
#define DST_REG_CAN1_RX_D47_32        (DST_CAN1_BASE + 0x36)  /* .... */
#define DST_REG_CAN1_RX_D31_16        (DST_CAN1_BASE + 0x38)  /* .... */
#define DST_REG_CAN1_RX_D15_00        (DST_CAN1_BASE + 0x3A)  /* Lo Data bits */
#define DST_REG_CAN1_RX_LEN           (DST_CAN1_BASE + 0x3C)  /* Length */
#define DST_REG_CAN1_RX_FLAGS         (DST_CAN1_BASE + 0x3E)  /* Control */

/* Status/IRQ registers */
#define DST_REG_CAN1_TX_ERR_CNT       (DST_CAN1_BASE + 0x40)  /* Rx/Tx err count  */
#define DST_REG_CAN1_ERR_STAT_CNT     (DST_CAN1_BASE + 0x42)  /* Error Stat count */
#define DST_REG_CAN1_MSG_LEVEL        (DST_CAN1_BASE + 0x44)  /* Msg Level Thresh */
#define DST_REG_CAN1_IRQ_FLAGS        (DST_CAN1_BASE + 0x46)  /* Interrupt Flags  */
#define DST_REG_CAN1_IRQ_ENABLE       (DST_CAN1_BASE + 0x48)  /* Interrupt enable */

/* CAN Configuration */
#define DST_REG_CAN1_MODE             (DST_CAN1_BASE + 0x4A)  /* Operating mode */
#define DST_REG_CAN1_BITRATE          (DST_CAN1_BASE + 0x4C)  /* */
#define DST_REG_CAN1_TIMING           (DST_CAN1_BASE + 0x4E)  /* */

/* Acceptance mask and code registers */
#define DST_REG_CAN1_FILTER_ENA       (DST_CAN1_BASE + 0x50)  /* Filter enables */

#define DST_REG_CAN1_AMR0_ID28_13     (DST_CAN1_BASE + 0x52)  /*Acceptance mask 0*/
#define DST_REG_CAN1_AMR0_ID12_00     (DST_CAN1_BASE + 0x54)  /* .. */
#define DST_REG_CAN1_AMR0_D63_48      (DST_CAN1_BASE + 0x56)  /* .. */
#define DST_REG_CAN1_ACR0_ID28_13     (DST_CAN1_BASE + 0x58)  /* Acceptance code 0*/
#define DST_REG_CAN1_ACR0_ID12_00     (DST_CAN1_BASE + 0x5A)  /* .. */
#define DST_REG_CAN1_ACR0_D63_48      (DST_CAN1_BASE + 0x5C)  /* .. */

#define DST_REG_CAN1_AMR1_ID28_13     (DST_CAN1_BASE + 0x5E)  /*Acceptance mask 1*/
#define DST_REG_CAN1_AMR1_ID12_00     (DST_CAN1_BASE + 0x60)  /* .. */
#define DST_REG_CAN1_AMR1_D63_48      (DST_CAN1_BASE + 0x62)  /* .. */
#define DST_REG_CAN1_ACR1_ID28_13     (DST_CAN1_BASE + 0x64)  /* Acceptance code 1*/
#define DST_REG_CAN1_ACR1_ID12_00     (DST_CAN1_BASE + 0x66)  /* .. */
#define DST_REG_CAN1_ACR1_D63_48      (DST_CAN1_BASE + 0x68)  /* .. */

#define DST_REG_CAN1_AMR2_ID28_13     (DST_CAN1_BASE + 0x6A)  /*Acceptance mask 2*/
#define DST_REG_CAN1_AMR2_ID12_00     (DST_CAN1_BASE + 0x6C)  /* .. */
#define DST_REG_CAN1_AMR2_D63_48      (DST_CAN1_BASE + 0x6E)  /* .. */
#define DST_REG_CAN1_ACR2_ID28_13     (DST_CAN1_BASE + 0x70)  /* Acceptance code 2*/
#define DST_REG_CAN1_ACR2_ID12_00     (DST_CAN1_BASE + 0x72)  /* .. */
#define DST_REG_CAN1_ACR2_D63_48      (DST_CAN1_BASE + 0x74)  /* .. */

/* Analysis registers */
#define DST_REG_CAN1_ALCR             (DST_CAN1_BASE + 0x76)  /* Arb. Lost Cap Reg*/
#define DST_REG_CAN1_ECR              (DST_CAN1_BASE + 0x78)  /* Error Capture Reg*/
#define DST_REG_CAN1_FRR              (DST_CAN1_BASE + 0x7A)  /* Frame Ref. Reg. */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Serial Peripheral Inteface registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_SPI0_BASE          0xB800

#define DST_REG_SPICTRHI      (DST_SPI0_BASE + 0x0A)   /* High byte of counter */
#define DST_REG_SPICTRLO      (DST_SPI0_BASE + 0x08)   /* Low byte of counter */
#define DST_REG_SPISSEL       (DST_SPI0_BASE + 0x06)   /* Slave Select */
```

```
#define DST_REG_SPISTAT       (DST_SPI0_BASE + 0x04)   /* Status */
#define DST_REG_SPICTRL       (DST_SPI0_BASE + 0x02)   /* Control */
#define DST_REG_SPIDATA       (DST_SPI0_BASE + 0x00)   /* Write / Read data */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** USB Controller Registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_USB0_BASE                0x9800

#define DST_REG_USB_INT_STAT         (DST_USB0_BASE + 0x00) /* Interrupt status */
#define DST_REG_USB_INT_ENB          (DST_USB0_BASE + 0x01) /* Interrupt enable */
#define DST_REG_USB_ERR_STAT         (DST_USB0_BASE + 0x02) /* Error Int. status */
#define DST_REG_USB_ERR_ENB          (DST_USB0_BASE + 0x03) /* Error Int. enable */
#define DST_REG_USB_STAT             (DST_USB0_BASE + 0x04) /* Status */
#define DST_REG_USB_CTL              (DST_USB0_BASE + 0x05) /* Control */
#define DST_REG_USB_ADDR             (DST_USB0_BASE + 0x06) /* Address */
#define DST_REG_USB_BDT_PAGE         (DST_USB0_BASE + 0x07) /* Buf descr table */
#define DST_REG_USB_FRM_NUML  (DST_USB0_BASE + 0x08) /* Frame num 7:0 */
#define DST_REG_USB_FRM_NUMH  (DST_USB0_BASE + 0x09) /* Frame num 10:8 */
#define DST_REG_USB_TOKEN            (DST_USB0_BASE + 0x0A) /* Token */
#define DST_REG_USB_SOF_THLDL (DST_USB0_BASE + 0x0B) /* SOF Thres 15:8 */
#define DST_REG_USB_SOF_THLDH        (DST_USB0_BASE + 0x0C) /* SOF Threshold 7:0 */

#define DST_REG_USB_ENDPT0    (DST_USB0_BASE + 0x10) /* Endpoint 0 control */
#define DST_REG_USB_ENDPT1    (DST_USB0_BASE + 0x11) /* Endpoint 1 control */
#define DST_REG_USB_ENDPT2    (DST_USB0_BASE + 0x12) /* Endpoint 2 control */
#define DST_REG_USB_ENDPT3    (DST_USB0_BASE + 0x13) /* Endpoint 3 control */
#define DST_REG_USB_ENDPT4    (DST_USB0_BASE + 0x14) /* Endpoint 4 control */
#define DST_REG_USB_ENDPT5    (DST_USB0_BASE + 0x15) /* Endpoint 5 control */
#define DST_REG_USB_ENDPT6    (DST_USB0_BASE + 0x16) /* Endpoint 6 control */
#define DST_REG_USB_ENDPT7    (DST_USB0_BASE + 0x17) /* Endpoint 7 control */
#define DST_REG_USB_ENDPT8    (DST_USB0_BASE + 0x18) /* Endpoint 8 control */
#define DST_REG_USB_ENDPT9    (DST_USB0_BASE + 0x19) /* Endpoint 9 control */
#define DST_REG_USB_ENDPT10   (DST_USB0_BASE + 0x1A) /* Endpoint 10 control */
#define DST_REG_USB_ENDPT11   (DST_USB0_BASE + 0x1B) /* Endpoint 11 control */
#define DST_REG_USB_ENDPT12   (DST_USB0_BASE + 0x1C) /* Endpoint 12 control */
#define DST_REG_USB_ENDPT13   (DST_USB0_BASE + 0x1D) /* Endpoint 13 control */
#define DST_REG_USB_ENDPT14   (DST_USB0_BASE + 0x1E) /* Endpoint 14 control */
#define DST_REG_USB_ENDPT15   (DST_USB0_BASE + 0x1F) /* Endpoint 15 control */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** I2C Controller Registers
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_I2C0_BASE         0xD000

#define DST_REG_I2CSRST       (DST_I2C0_BASE + 0x0E)   /* Software Reset */
#define DST_REG_I2CXADDR      (DST_I2C0_BASE + 0x08)   /* Extended Slave Addr */
#define DST_REG_I2CCCR        (DST_I2C0_BASE + 0x06)   /* Clock Control */
#define DST_REG_I2CSTAT       (DST_I2C0_BASE + 0x06)   /* Status */
#define DST_REG_I2CCNTR       (DST_I2C0_BASE + 0x04)   /* Control */
#define DST_REG_I2CDATA       (DST_I2C0_BASE + 0x02)   /* Data */
#define DST_REG_I2CADDR       (DST_I2C0_BASE + 0x00)   /* Address */
```

```
/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Register field defines and masks
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/* PCB/RELREG - Relocation Register */
#define DST_RELREG_PCB_IN_MEM         0x1000   /* Puts PCB in memory space */
#define DST_RELREG_PCB_IN_IO                  0x0000   /* Puts PCB in I/O space */

/* PRL - Processor Revision Level */
#define DST_PRL_MSK_REV                       0xFF00   /* Revision Number */
#define DST_PRL_MSK_CPU                       0x00FF   /* CPU number */

/* PCB/AUXCON - Auxiliary Configuration */
#define DST_AUXCON_ENRX3                      0x0400   /* Selects ENRX mode SP3 CTS/ENRX */
#define DST_AUXCON_DTE3                       0x0200   /* Selects RTS mode SP3 RTR/RTS */
#define DST_AUXCON_ENRX2                      0x0100   /* Selects ENRX mode SP2 CTS/ENRX */
#define DST_AUXCON_DTE2                       0x0080   /* Selects RTS mode SP2 RTR/RTS pin */
#define DST_AUXCON_ENRX1                      0x0040   /* Selects ENRX mode SP1 CTS/ENRX  */
#define DST_AUXCON_DTE1                       0x0020   /* Selects RTS mode SP1 RTR/RTS pin */
#define DST_AUXCON_ENRX0                      0x0010/* Selects ENRX mode SP0 CTS/ENRX pin */
#define DST_AUXCON_DTE0                       0x0008   /* Selects RTS mode SP0 RTR/RTS pin */

/* PCB/SYSCON - System Configuration (read-only) */
#define DST_SYSCON_MCS0               0x4000  /* Enable MSC0 over entire MCS range */
#define DST_SYSCON_CD                 0x0100  /* Disable clock output */

/* DCR - DSTni Configuration Register */
#define DST_DCR_BROMEN                0x4000  /* Boot ROM enabled */
#define DST_DCR_ADDR24                0x2000  /* Extended address mode enabled */
#define DST_DCR_WDOGEN        0x1000  /* Watchdog enabled */
#define DST_DCR_SPIBOOT                       0x0800  /* Boot from SPI enabled */
#define DST_DCR_SPIEN                 0x0400  /* SPI pins enabled in PIO */
#define DST_DCR_ETHBOOT               0x0200  /* Ethernet boot enabled */
#define DST_DCR_ETHCHAN               0x0100  /* Ethernet channel 1 selected */
#define DST_DCR_BICOLOR                       0x0080  /* Using encoded LEDs */
#define DST_DCR_PARBOOT               0x0040  /* Boot from parallel flash enabled */
#define DST_DCR_PHYBYPASS             0x0020  /* Do not access Ethernet PHY via MII */
#define DST_DCR_BYTEMODE              0x0010  /* Select hi/lo byte read */
#define DST_DCR_SERBOOT               0x0008  /* Enable boot via serial port */
#define DST_DCR_SERCHAN               0x0004  /* Select serial port 0 or 1 for boot */
#define DST_DCR_SERSPEED              0x0002  /* Select 9600 baud serial speed */
#define DST_DCR_DEBUG                 0x0001  /* Enable boot debug messages */

/* PCB/WDTCON - Watchdog control */
#define DST_WDTCON_ENA                        0x8000   /* Enable timer */
#define DST_WDTCON_WRST                       0x4000   /* Cause a reset upon timeout */
#define DST_WDTCON_RSTFLAG            0x2000   /* Set if a WD reset has happened */
#define DST_WDTCON_NMIFLAG            0x1000   /* Set if a NMI event occurred */
#define DST_WDTCON_UNLOCK             0x0800   /* Unlock reg for further writes */
#define DST_WDTCON_EXP_10                     0x0001   /* Set WDT timer exponent to 10 */
#define DST_WDTCON_EXP_20                     0x0002   /* Set WDT timer exponent to 20 */
#define DST_WDTCON_EXP_21                     0x0004   /* Set WDT timer exponent to 21 */
#define DST_WDTCON_EXP_22                     0x0008   /* Set WDT timer exponent to 22 */
#define DST_WDTCON_EXP_23                     0x0010   /* Set WDT timer exponent to 23 */
#define DST_WDTCON_EXP_24                     0x0020   /* Set WDT timer exponent to 24 */
#define DST_WDTCON_EXP_25                     0x0040   /* Set WDT timer exponent to 25 */
#define DST_WDTCON_EXP_26                     0x0080   /* Set WDT timer exponent to 26 */
#define DST_WDTCON_EXP_27                     0x0100   /* Set WDT timer exponent to 27 */
#define DST_WDTCON_EXP_28                     0x0200   /* Set WDT timer exponent to 28 */
#define DST_WDTCON_EXP_29                     0x0400   /* Set WDT timer exponent to 29 */
#define DST_WDTCON_RESET_1            0xAAAA   /* First of 2 reset steps */
#define DST_WDTCON_RESET_2            0x5555   /* Second of 2 reset steps */
#define DST_WDTCON_OPEN_1                     0x3333   /* First of 2 open-for-write steps */
#define DST_WDTCON_OPEN_2                     0xCCCC   /* Second of 2 open-for-write steps */
```

```
/* PCB/T0CON - Timer 0 Control */
#define DST_T0CON_ENABLE                      0xC000   /* Enables Timer */
#define DST_T0CON_DISABLE                     0x4000   /* Disables Timer */
#define DST_T0CON_INT_ENABLE          0x2000   /* Enable interrupts */
#define DST_T0CON_RIU                         0x1000   /* 0 = maxcount A in use,else maxcnt B*/
#define DST_T0CON_MC                          0x0020   /* If 1, maxcount has been reached */
#define DST_T0CON_RTG                         0x0010   /* Use TMRIN0 to reset count */
#define DST_T0CON_P                           0x0008   /* Use Timer 2 as prescale */
#define DST_T0CON_EXT                         0x0004   /* Use TMRIN0 as external clock source */
#define DST_T0CON_ALT                         0x0002   /* Alternate between Maxcount A and B */
#define DST_T0CON_CONT                        0x0001   /* Run in continuous mode */

/* PCB/T1CON - Timer 1 Control */
#define DST_T1CON_ENABLE                      0xC000   /* Enables Timer */
#define DST_T1CON_DISABLE                     0x4000   /* Disables Timer */
#define DST_T1CON_INT_ENABLE          0x2000   /* Enable interrupts */
#define DST_T1CON_RIU                         0x1000   /* 0 = maxcnt A in use, else maxcnt B*/
#define DST_T1CON_MC                          0x0020   /* If 1, maxcnt has been reached */
#define DST_T1CON_RTG                         0x0010   /* Use TMRIN0 to reset count */
#define DST_T1CON_P                           0x0008   /* Use Timer 2 as prescale */
#define DST_T1CON_EXT                         0x0004   /* Use TMRIN0 as external clock source */
#define DST_T1CON_ALT                         0x0002   /* Alternate between Maxcnt A and B */
#define DST_T1CON_CONT                        0x0001   /* Run in continuous mode */

/* PCB/T2CON - Timer 2 Control */
#define DST_T2CON_ENABLE                      0xC000   /* Enables Timer */
#define DST_T2CON_DISABLE                     0x4000   /* Disables Timer */
#define DST_T2CON_INT_ENABLE          0x2000   /* Enable interrupts */
#define DST_T2CON_MC                          0x0020   /* If 1, maxcount has been reached */
#define DST_T2CON_CONT                        0x0001   /* Run in continuous mode */

/* PCB/INSERV - Interrupt in-service */
#define DST_INSERV_CAN                        0x4000   /* I6: CAN1 and CAN0 */
#define DST_INSERV_DMA3                       0x2000   /* DMA channel 3 */
#define DST_INSERV_DMA2                       0x1000   /* DMA channel 2 */
#define DST_INSERV_SP3                        0x0800   /* I5: serial port 3 */
#define DST_INSERV_INT5                       0x0800   /* I5: Extneral INT5 pin */
#define DST_INSERV_SP0                        0x0400   /* Serial Port 0 */
#define DST_INSERV_SP1                        0x0200   /* Serial Port 1 */
#define DST_INSERV_SP2                        0x0100   /* Serial Port 2 */
#define DST_INSERV_USB                        0x0080   /* I3: USB */
#define DST_INSERV_INT3                       0x0080   /* I3: External INT 3 pin */
#define DST_INSERV_SPI                        0x0040   /* I2: SPI */
#define DST_INSERV_I2C                        0x0040   /* I2: I2C */
#define DST_INSERV_MAC1                       0x0020   /* I1: Ethernet MAC1 */
#define DST_INSERV_INT1                       0x0020   /* I1: External INT1 pin */
#define DST_INSERV_MAC0                       0x0010   /* I0: Ethernet MAC0 */
#define DST_INSERV_DMA1                       0x0008   /* DMA channel 1 */
#define DST_INSERV_DMA0                       0x0004   /* DMA channel 0 */
#define DST_INSERV_TMR                        0x0001   /* Timer (0, 1 or 2) */

/* PCB/REQST - Interrupt request */
#define DST_REQST_DMA3                0x2000   /* DMA channel 3 */
#define DST_REQST_DMA2                0x1000   /* DMA channel 2 */
#define DST_REQST_SP3                 0x0800   /* I5: serial port 3 */
#define DST_REQST_INT5                        0x0800   /* I5: Extneral INT5 pin */
#define DST_REQST_SP0                 0x0400   /* Serial Port 0 */
#define DST_REQST_SP1                 0x0200   /* Serial Port 1 */
#define DST_REQST_SP2                 0x0100   /* Serial Port 2 */
#define DST_REQST_USB                 0x0080   /* I3: USB */
#define DST_REQST_INT3                0x0080   /* I3: External INT 3 pin */
#define DST_REQST_SPI                 0x0040   /* I2: SPI */
#define DST_REQST_I2C                 0x0040   /* I2: I2C */
#define DST_REQST_INT1                0x0020   /* I1: External INT1 pin */
#define DST_REQST_MAC1                        0x0020   /* I1: Ethernet MAC1 */
#define DST_REQST_MAC0                0x0010   /* I0: Ethernet MAC0 */
#define DST_REQST_DMA1                0x0008   /* DMA channel 1 */
#define DST_REQST_DMA0                0x0004   /* DMA channel 0 */
#define DST_REQST_TMR                 0x0001   /* Timer (0, 1 or 2) */
```

```
/* PCB/IMASK - Interrupt mask */
#define DST_IMASK_DMA3                0x2000   /* DMA channel 3 */
#define DST_IMASK_DMA2                0x1000   /* DMA channel 2 */
#define DST_IMASK_SP3                 0x0800   /* I5: serial port 3 */
#define DST_REQST_INT5                       0x0800   /* I5: Extneral INT5 pin */
#define DST_IMASK_SP0                 0x0400   /* Serial Port 0 */
#define DST_IMASK_SP1                 0x0200   /* Serial Port 1 */
#define DST_IMASK_SP2                 0x0100   /* Serial Port 2 */
#define DST_IMASK_USB                 0x0080   /* I3: USB */
#define DST_IMASK_INT3                0x0080   /* I3: External INT 3 pin */
#define DST_IMASK_SPI                 0x0040   /* I2: SPI */
#define DST_IMASK_I2C                 0x0040   /* I2: I2C */
#define DST_IMASK_INT1                0x0020   /* I1: External INT1 pin */
#define DST_IMASK_MAC1                       0x0020   /* I1: Ethernet MAC1 */
#define DST_IMASK_MAC0                0x0010   /* I0: Ethernet MAC0 */
#define DST_IMASK_DMA1                0x0008   /* DMA channel 1 */
#define DST_IMASK_DMA0                0x0004   /* DMA channel 0 */
#define DST_IMASK_TMR                 0x0001   /* Timer (0, 1 or 2) */
#define DST_IMASK_ALL                        0x7FFD   /* Mask all interrupt sources */

/* PCB/PRIMSK - Interrupt priority mask */
#define DST_PRIMSK_L7                        0x0007   /* Enable all priority levels (0 - 7) */
#define DST_PRIMSK_L6                        0x0006   /* Enable levels 0 through 6 */
#define DST_PRIMSK_L5                        0x0005   /* Enable levels 0 through 5 */
#define DST_PRIMSK_L4                        0x0004   /* Enable levels 0 through 4 */
#define DST_PRIMSK_L3                        0x0003   /* Enable levels 0 through 3 */
#define DST_PRIMSK_L2                        0x0002   /* Enable levels 0 through 2 */
#define DST_PRIMSK_L1                        0x0001   /* Enable levels 0 through 1 */
#define DST_PRIMSK_L0                        0x0000   /* Enable level 0 */

/* PCB/INSTS - Interrupt status */
#define DST_INTSTS_DHLT                      0x8000   /* Halts DMA activity when set */
#define DST_INTSTS_TMR2                      0x0004   /* Timer 2 is requesting an interrupt */
#define DST_INTSTS_TMR1                      0x0002   /* Timer 1 is requesting an interrupt */
#define DST_INTSTS_TMR0                      0x0001   /* Timer 0 is requesting an interrupt */

/* PCB/xxxCON - Peripheral interrupt control (timer, serial, dma, spi) */
#define DST_PERCON_MSK                       0x0008   /* Mask this interrupt */
#define DST_PERCON_PRI_L7                    0x0007   /* Set interrupt priority 7 (lowest) */
#define DST_PERCON_PRI_L6                    0x0006   /* Set interrupt priority 6 */
#define DST_PERCON_PRI_L5                    0x0005   /* Set interrupt priority 5 */
#define DST_PERCON_PRI_L4                    0x0004   /* Set interrupt priority 4 */
#define DST_PERCON_PRI_L3                    0x0003   /* Set interrupt priority 3 */
#define DST_PERCON_PRI_L2                    0x0002   /* Set interrupt priority 2 */
#define DST_PERCON_PRI_L1                    0x0001   /* Set interrupt priority 1 */
#define DST_PERCON_PRI_L0                    0x0000   /* Set interrupt priority 0 (highest) */

/* PCB/IxCON - Interrupt 5-0 control */
#define DST_IXCON_LTM                        0x0010   /* Select level triggering */
#define DST_IXCON_MSK                        0x0008   /* Mask this interrupt */
#define DST_IXCON_PRI_L7                     0x0007   /* Set interrupt priority 7 (lowest) */
#define DST_IXCON_PRI_L6                     0x0006   /* Set interrupt priority 6 */
#define DST_IXCON_PRI_L5                     0x0005   /* Set interrupt priority 5 */
#define DST_IXCON_PRI_L4                     0x0004   /* Set interrupt priority 4 */
#define DST_IXCON_PRI_L3                     0x0003   /* Set interrupt priority 3 */
#define DST_IXCON_PRI_L2                     0x0002   /* Set interrupt priority 2 */
#define DST_IXCON_PRI_L1                     0x0001   /* Set interrupt priority 1 */
#define DST_IXCON_PRI_L0                     0x0000   /* Set interrupt priority 0 (highest) */

/* PCB/EOI - End of interrupt */
#define DST_EOI_NONSPEC                      0x8000   /* Non-specific end of interrupt */

/* PCB/POLL/POLLST - Interrupt Poll and Poll Status */
#define DST_POLL_IRQ                         0x8000   /* Set if interrupt is pending */
#define DST_POLL_VECTOR_MSK           0x001F   /* Mask off the interrupt vector */
```

```
/* Interrupt vectors - devices */
#define DST_IVECT_CAN                          24       /* I6: CAN1 & CAN0 */
#define DST_IVECT_DMA3                         23       /* DMA channel 3 */
#define DST_IVECT_DMA2                         22       /* DMA channel 2 */
#define DST_IVECT_SP3                          21       /* I5: Serial port 3 */
#define DST_IVECT_INT5                         21       /* I5: External INT5 pin */
#define DST_IVECT_SP0                          20       /* Serial Port 0 */
#define DST_IVECT_TMR2                         19       /* Timer 2 */
#define DST_IVECT_TMR1                         18       /* Timer 1 */
#define DST_IVECT_SP1                          17       /* Serial Port 1 */
#define DST_IVECT_SP2                          16       /* Serial port 2 */
#define DST_IVECT_USB                          15       /* I3: USB controller */
#define DST_IVECT_INT3                         15       /* I3: External INT3 pin */
#define DST_IVECT_SPI                          14       /* I2: SPI controller */
#define DST_IVECT_I2C                          14       /* I2: I2C controller */
#define DST_IVECT_INT1                         13       /* I1: External INT1 pin */
#define DST_IVECT_MAC1                         13       /* I1: Ethernet MAC1 */
#define DST_IVECT_MAC0                         12       /* INT0: Ethernet MAC0 */
#define DST_IVECT_DMA1                         11       /* DMA channel 1 */
#define DST_IVECT_DMA0                         10       /* DMA channel 0 */
#define DST_IVECT_TMR0                         8        /* Timer 0 */

/* Interrupt vectors - software */
#define DST_IVECT_DIVIDE                       0        /* Divide error */
#define DST_IVECT_SSTEP                        1        /* Single Step */
#define DST_IVECT_NMI                          2        /* Non-Maskable Interrupt */
#define DST_IVECT_BREAKPT                      3        /* Software breakpoint */
#define DST_IVECT_INTO                         4        /* INTO instruction + OF flag set */
#define DST_IVECT_BOUNDS                       5        /* Array bounds exception */
#define DST_IVECT_INV_OP                       6        /* Unused opcode */
#define DST_IVECT_ESC_OP                       7        /* Escape (ESC) trap */

/* PCB/DMA - DMA control */
#define DST_DMA_DST_MEM                        0x8000   /* Destination is in memory space */
#define DST_DMA_DST_IO                         0x0000   /* Destination is in I/O space */
#define DST_DMA_DST_DEC                        0x4000   /* Decrement dst pointer */
#define DST_DMA_DST_INC                        0x2000   /* Increment dst pointer */
#define DST_DMA_SRC_MEM                        0x1000   /* Source is in memory space */
#define DST_DMA_SRC_IO                         0x0000   /* Source is in I/O space */
#define DST_DMA_SRC_DEC                        0x0800   /* Decrement src pointer */
#define DST_DMA_SRC_INC                        0x0400   /* Increment src pointer */
#define DST_DMA_TC                             0x0200   /* Stop DMA when count reaches 0 */
#define DST_DMA_INT_ENABLE                     0x0100   /* Generate interrupt on termination */
#define DST_DMA_SYNC_NO                        0x0000   /* No synchronization */
#define DST_DMA_SYNC_DST                       0x0080   /* Destination sync */
#define DST_DMA_SYNC_SRC                       0x0040   /* Source sync */
#define DST_DMA_PRI_HI                         0x0020   /* Select high priority */
#define DST_DMA_PRI_LO                         0x0000   /* Select low priority */
#define DST_DMA_TDRQ                           0x0010   /* Timer 2 provides request */
#define DST_DMA_START                          0x0006   /* Start DMA operation */
#define DST_DMA_STOP                           0x0004   /* Stop DMA operation */
#define DST_DMA_XFER_WORD               0x0001   /* Select word transfers */
#define DST_DMA_XFER_BYTE                      0x0000   /* Select byte transfers */
```

```
/* PCB/SPxCT - Serial port control */
#define DST_SPXCT_DMA_MODE_0          0x0000   /* DMA Mode 0 */
#define DST_SPXCT_DMA_MODE_1  0x2000   /* DMA Mode 1 */
#define DST_SPXCT_DMA_MODE_2          0x4000   /* DMA Mode 2 */
                                          /* No DMA mode 3 */
#define DST_SPXCT_DMA_MODE_4          0x8000   /* DMA Mode 4 */
#define DST_SPXCT_DMA_MODE_5          0xA000   /* DMA Mode 5 */
#define DST_SPXCT_DMA_MODE_6          0xC000   /* DMA Mode 6 */
#define DST_SPXCT_DMA_MODE_7          0xE000   /* DMA Mode 7 */
#define DST_SPXCT_RSIE                        0x1000   /* Receive Status Interrupt Enable */
#define DST_SPXCT_BRK                         0x0800   /* Generate Break */
#define DST_SPXCT_TB8                         0x0400   /* Transmit Bit 8 */
#define DST_SPXCT_FC_EN                       0x0200   /* Enable flow control */
#define DST_SPXCT_TXIE                        0x0100   /* Transmitter empty interrupt enable */
#define DST_SPXCT_RXIE                        0x0080   /* Receive data interrupt enable */
#define DST_SPXCT_TMOD                        0x0040   /* Enable transmitter */
#define DST_SPXCT_RMOD                        0x0020   /* Enable receiver */
#define DST_SPXCT_PARITY_EVN                  0x0018   /* Select even parity */
#define DST_SPXCT_PARITY_ODD          0x0008   /* Select odd parity */
#define DST_SPXCT_PARITY_NONE         0x0000   /* Select no parity */
#define DST_SPXCT_MODE_1                      0x0001   /* Select mode 1 */
#define DST_SPXCT_MODE_2                      0x0002   /* Select mode 2 */
#define DST_SPXCT_MODE_3                      0x0003   /* Select mode 3 */
#define DST_SPXCT_MODE_4                      0x0004   /* Select mode 4 */
#define DST_SPXCT_MODE_5                      0x0005   /* Select mode 4 */

/* PCB/SPxSTS - Serial port status */
#define DST_SPXSTS_BRK1                       0x0400   /* Long break detected */
#define DST_SPXSTS_BRK0                       0x0200   /* Short break detected */
#define DST_SPXSTS_RB8                        0x0100   /* Receive bit 8 */
#define DST_SPXSTS_RDR                        0x0080   /* Receive data ready */
#define DST_SPXSTS_THRE                       0x0040   /* Transmit holding register empty */
#define DST_SPXSTS_FER                        0x0020   /* Framing error */
#define DST_SPXSTS_OER                        0x0010   /* Overrun error */
#define DST_SPXSTS_PER                        0x0008   /* Parity error */
#define DST_SPXSTS_TEMT                       0x0004   /* Transmitter empty */
#define DST_SPXSTS_HS0                        0x0002   /* Handshake 0 (CTS) active */

/* PCB/SPxAUX - Serial port Aux control */
#define DST_SPXAUX_BRGO               0x0800   /* Connect RTS out to baud generator */
#define DST_SPXAUX_FIFO_1             0x0000   /* Set FIFO depth to 1 byte */
#define DST_SPXAUX_FIFO_2             0x0020   /* Set FIFO depth to 2 bytes */
#define DST_SPXAUX_FIFO_3             0x0040   /* Set FIFO depth to 3 bytes */
#define DST_SPXAUX_FIFO_4             0x0060   /* Set FIFO depth to 4 bytes */
#define DST_SPXAUX_RTSZ                       0x0010   /* Forcr RTS inactive */
#define DST_SPXAUX_RTS                0x0008   /* Force RTS active */
#define DST_SPXAUX_RXM                0x0004   /* Force receiver to half-duplex */
#define DST_SPXAUX_CTSM               0x0002   /* Force CTS active internally */
#define DST_SPXAUX_RTSP               0x0001   /* Invert RTS polarity */

/* PCB/SPICTRL - Serial peripheral interface control */
#define DST_SPICTRL_IRQEN                     0x0080   /* Enable interrupt */
#define DST_SPICTRL_AUTODRV                   0x0040   /* Enable Autodrv */
#define DST_SPICTRL_INVCS                     0x0020   /* Invert Chip Select */
#define DST_SPICTRL_PHASE_0                   0x0000   /* Select Phase 0 */
#define DST_SPICTRL_PHASE_1                   0x0010   /* Select Phase 1 */
#define DST_SPICTRL_CKPOL_HI                  0x0008   /* Select 'high' clock idle */
#define DST_SPICTRL_CKPOL_LO                  0x0000   /* Select 'low' clock idle */
#define DST_SPICTRL_WOR_EN                    0x0004   /* Enable wire-or operation */
#define DST_SPICTRL_MSTEN                     0x0002   /* Assert mastery of bus */
#define DST_SPICTRL_ALT               0x0001   /* Select alternate I/O pinout */

/* PCB/SPISTAT - Serial peripheral interface status */
#define DST_SPISTAT_IRQ                       0x0080   /* Interrupt has occurred */
#define DST_SPISTAT_OVERRUN                   0x0040   /* Transmit overrun */
#define DST_SPISTAT_COL                       0x0020   /* Collision between bus masters */
#define DST_SPISTAT_TXRUN                     0x0002   /* Master mode operation in progress */
#define DST_SPISTAT_SLVSEL                    0x0001   /* External master is active on bus */
```

```
/* PCB/SPISSEL - Serial peripheral interface slave select */
#define DST_SPISSEL_SHIFT_8                   0x0000   /* Select 8 bit transfer */
#define DST_SPISSEL_SHIFT_7                   0x00E0   /* Select 7 bit transfer */
#define DST_SPISSEL_SHIFT_6                   0x00C0   /* Select 6 bit transfer */
#define DST_SPISSEL_SHIFT_5                   0x00A0   /* Select 5 bit transfer */
#define DST_SPISSEL_SHIFT_4                   0x0080   /* Select 4 bit transfer */
#define DST_SPISSEL_SHIFT_3                   0x0060   /* Select 3 bit transfer */
#define DST_SPISSEL_SHIFT_2                   0x0040   /* Select 2 bit transfer */
#define DST_SPISSEL_SHIFT_1                   0x0020   /* Select 1 bit transfer */
#define DST_SPISSEL_SELECTO                   0x0001   /* Drive SLVSEL pin active */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Ethernet Controller Constants
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
/* Ethernet/RAP - Register Address Port */
/* Here are the registers accessible via RAP/RDP */
#define DST_ETHRAP_STATUS                     0        /* CSR0 - Status */
#define DST_ETHRAP_IADR_L                     1        /* CSR1 - initialization addr low word */
#define DST_ETHRAP_IADR_H                     2        /* CSR2 - initialization addr high word*/
#define DST_ETHRAP_INTMSK                     3        /* CSR3 - interrupt mask */
#define DST_ETHRAP_FEATURE                    4        /* CSR4 - features control */
#define DST_ETHRAP_PADR_15_00        12       /* CSR12 - MAC address bits 15-0 */
#define DST_ETHRAP_PADR_31_16        13       /* CSR13 - MAC address bits 31-16 */
#define DST_ETHRAP_PADR_47_32        14       /* CSR14 - MAC address bits 47-32 */
#define DST_ETHRAP_MODE                       15       /* CSR15 - Mode control */
#define DST_ETHRAP_RXBASE_L                   24       /* CSR24 - RX ring base addr low word */
#define DST_ETHRAP_RXBASE_H          25       /* CSR25 - RX ring base addr high word */
#define DST_ETHRAP_TXBASE_L                   30       /* CSR30 - TX ring base addr low word */
#define DST_ETHRAP_TXBASE_H                   31       /* CSR31 - TX ring base addr high word */
#define DST_ETHRAP_POLL_TIME                  46       /* CSR46 - Poll time counter */
#define DST_ETHRAP_POLL_INTVL        47       /* CSR47 - Poll interval */
#define DST_ETHRAP_RXLEN                      76       /* CSR76 - Receive ring length */
#define DST_ETHRAP_TXLEN                      78       /* CSR78 - Transmit length */
#define DST_ETHRAP_CHIP_ID_L                  88       /* CSR88 - Chip ID low word */
#define DST_ETHRAP_CHIP_ID_H                  89       /* CSR89 - Chip ID high word */
#define DST_ETHRAP_MISSD_FRM         112      /* CSR112 - Missed frame count */
#define DST_ETHRAP_RCV_COL                    114      /* CSR114 - Received collision count */

/* Ethernet Status register bits (CSR0) */
#define DST_ETHSTAT_ERR                       0x8000   /* Logical or of CERR, MISS */
#define DST_ETHSTAT_CERR                      0x2000   /* SQE test error */
#define DST_ETHSTAT_MISS                      0x1000   /* Incoming frame was lost */
#define DST_ETHSTAT_RINT                      0x0400   /* Frame received in RX ring */
#define DST_ETHSTAT_TINT                      0x0200   /* Frame has been transmitted */
#define DST_ETHSTAT_IDON                      0x0100   /* Initialization block read */
#define DST_ETHSTAT_INSTR                     0x0080   /* Logical or of MISS, MFCO, RCVCCO, */
    /* RINT, TINT, IDON, TXSTRT, PAUSE */
#define DST_ETHSTAT_IENA                      0x0040   /* Interrupt enable */
#define DST_ETHSTAT_RXON                      0x0020   /* Enable receiver */
#define DST_ETHSTAT_TXON                      0x0010   /* Enable transmitter */
#define DST_ETHSTAT_TDMD                      0x0008   /* Force poll of RX & TX rings */
#define DST_ETHSTAT_STOP                      0x0004   /* Stop activity */
#define DST_ETHSTAT_STRT                      0x0002   /* Start activity */
#define DST_ETHSTAT_INIT                      0x0001   /* Load initialization block */

/* Ethernet Interrupt mask (CSR3) */
#define DST_ETHIMSK_MISSM                     0x1000   /* Mask missed frame interrupt */
#define DST_ETHIMSK_RINTM                     0x0400   /* Mask received frame interrupt */
#define DST_ETHIMSK_TINTM                     0x0200   /* Mask transmit done interrput */
#define DST_ETHIMSK_IDONM                     0x0100   /* Mask initialization done interrupt */
#define DST_ETHIMSK_DTX2PD                    0x0010   /* Disable transmit 2 part deferral */
```

```
/* Ethernet Features control (CSR4) */
#define DST_ETHFEAT_DPOLL                      0x1000   /* Disable transmit polling */
#define DST_ETHFEAT_APADTX                     0x0800   /* Pad frames shorter than 64 bytes */
#define DST_ETHFEAT_MFCO                       0x0200   /* Missed frame counter overflow */
#define DST_ETHFEAT_MFCOM                      0x0100   /* Mask MFCO interrupt */
#define DST_ETHFEAT_RPA                        0x0080   /* Enable runt packet reception */
#define DST_ETHFEAT_RCVCCO                     0x0020   /* Receive collision counter overflow */
#define DST_ETHFEAT_RCVCCOM            0x0010   /* Mask RCVCCO interrupt */
#define DST_ETHFEAT_TXSTRT                     0x0008   /* Transmit has started */
#define DST_ETHFEAT_TXSTRTM                    0x0004   /* Mask TXSTRT interrupt */
#define DST_ETHFEAT_PAUSE                      0x0002   /* Pause control frame received */
#define DST_ETHFEAT_PAUSEM                     0x0001   /* Mask PAUSE interrupt */

/* Ethernet Mode (CSR15) */
#define DST_ETHMODE_PROM                       0x8000   /* Enable promiscuous mode */
#define DST_ETHMODE_DRXBC                      0x4000   /* Disable multicast receive */
#define DST_ETHMODE_DRXPA                      0x2000   /* Disable receive physical address */
#define DST_ETHMODE_DPAUSE             0x0800   /* Disable automatic pause */
#define DST_ETHMODE_DRTY                       0x0020   /* Disable transmit retries */
#define DST_ETHMODE_DTXFCS             0x0008   /* Disable CRC generation */
#define DST_ETHMODE_DTX                        0x0002   /* Disable transmit operation */
#define DST_ETHMODE_DRX                        0x0001   /* Disable receive operation */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** Ethernet Media Independent Interface (MII) constants
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
#define DST_ETHMII_FDEN                        0x8000   /* Full-duplex enable */
#define DST_ETHMII_MDI                         0x0100   /* Management data in bit */
#define DST_ETHMII_MDOE                        0x0080   /* MDIO pin output enable */
#define DST_ETHMII_MDC                         0x0002   /* Management data clock */
#define DST_ETHMII_MDO                         0x0001   /* Management data out bit */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** CAN Register Constants
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
/* CAN Transmit message data length & control */
#define DST_CAN_TXMSG_IDE                      0x0010   /* Extended identifier bit */
#define DST_CAN_TXMSG_RTR                      0x0020   /* Remote bit */

/* CAN Transmit control flags */
#define DST_CAN_TXFL_TRX                       0x0001   /* Initiate transmit */
#define DST_CAN_TXFL_TXAB              0x0002   /* Abort transmit */

/* CAN Receive status masks */
#define DST_CAN_RXST_DLC               0x000F   /* Data length code */
#define DST_CAN_RXST_IDE               0x0010   /* Extended identifier bit */
#define DST_CAN_RXST_RTR               0x0020   /* Remote bit */
#define DST_CAN_RXST_AFI_0             0x0100   /* Acceptance filter 0 indicator */
#define DST_CAN_RXST_AFI_1             0x0200   /* Acceptance filter 1 indicator */
#define DST_CAN_RXST_AFI_2             0x0400   /* Acceptance filter 2 indicator */

/* CAN Error status register */
#define DST_CAN_ERST_STATE_BOFF        0x0002   /* Bus off, bit 0 is don't care */
#define DST_CAN_ERST_STATE_ACT         0x0000   /* Error active normal operation */
#define DST_CAN_ERST_STATE_PAS         0x0001   /* Error passive */
#define DST_CAN_ERST_TXGTE96           0x0004   /* Tx error count exceeds 96 */
#define DST_CAN_ERST_RXGTE96           0x0008   /* Rx error count exceeds 96 */

/* CAN Transmit Fifo interrupt levels */
#define DST_CAN_TXFIFO_LVL0            0x0000   /* Interrupt when all tx buffers MT */
#define DST_CAN_TXFIFO_LVL1            0x0001   /* . when at least 2 buffers empty */
#define DST_CAN_TXFIFO_LVL2            0x0002   /* . when at least 3 buffers empty */
```

```
/* CAN Receive Fifo interrupt levels */
#define DST_CAN_RXFIFO_LVL0           0x0000   /* Interrupt when at least 1 msg */
#define DST_CAN_RXFIFO_LVL1           0x0004   /* .. when at least 2 messages */
#define DST_CAN_RXFIFO_LVL2           0x0008   /* .. when at least 3 messages */
#define DST_CAN_RXFIFO_LVL3           0x000C   /* .. when at least 4 messages */

/* CAN Interrupt requests */
#define DST_CAN_IRQ_ARBLOSS           0x0004   /* Arbitration lost during tx */
#define DST_CAN_IRQ_OVRLOAD           0x0008   /* Overload condition */
#define DST_CAN_IRQ_RX_OVR            0x0010   /* Receiver overrun */
#define DST_CAN_IRQ_BIT_ERR           0x0020   /* Bit error during tx or rx */
#define DST_CAN_IRQ_STUF_ERR          0x0040   /* Stuffing error during tx or rx */
#define DST_CAN_IRQ_ACK_ERR           0x0080   /* Ack error during tx or rx */
#define DST_CAN_IRQ_FORM_ERR          0x0100   /* Format error during tx or rx */
#define DST_CAN_IRQ_CRC_ERR           0x0200   /* CRC error during tx or rx */
#define DST_CAN_IRQ_BUS_OFF           0x0400   /* CAN is in bus off state */
#define DST_CAN_IRQ_TX_XMIT0          0x0800   /* Message 0 sent */
#define DST_CAN_IRQ_TX_XMIT1          0x1000   /* Message 1 sent */
#define DST_CAN_IRQ_TX_XMIT2          0x2000   /* Message 2 sent */
#define DST_CAN_IRQ_TX_DONE           0x4000   /* At least 1 tx buffer empty */
#define DST_CAN_IRQ_RX_DONE           0x8000   /* At least 1 rx message available */

/* CAN Interrupt enable */
#define DST_CAN_IEN_GENRL             0x0001   /* General enable */
#define DST_CAN_IEN_ARBLOSS           0x0004   /* Arbitration lost during tx */
#define DST_CAN_IEN_OVRLOAD           0x0008   /* Overload condition */
#define DST_CAN_IEN_RX_OVR            0x0010   /* Receiver overrun */
#define DST_CAN_IEN_BIT_ERR           0x0020   /* Bit error during tx or rx */
#define DST_CAN_IEN_STUF_ERR          0x0040   /* Stuffing error during tx or rx */
#define DST_CAN_IEN_ACK_ERR           0x0080   /* Ack error during tx or rx */
#define DST_CAN_IEN_FORM_ERR          0x0100   /* Format error during tx or rx */
#define DST_CAN_IEN_CRC_ERR           0x0200   /* CRC error during tx or rx */
#define DST_CAN_IEN_BUS_OFF           0x0400   /* CAN is in bus off state */
#define DST_CAN_IEN_TX_XMIT0          0x0800   /* Message 0 sent */
#define DST_CAN_IEN_TX_XMIT1          0x1000   /* Message 1 sent */
#define DST_CAN_IEN_TX_XMIT2          0x2000   /* Message 2 sent */
#define DST_CAN_IEN_TX_DONE           0x4000   /* At least 1 tx buffer empty */
#define DST_CAN_IEN_RX_DONE           0x8000   /* At least 1 rx message available */

/* CAN operating mode */
#define DST_CAN_OPMOD_RUN             0x0001   /* Place controller to RUN mode */
#define DST_CAN_OPMOD_PAS             0x0002   /* Place controller to passive mode */
#define DST_CAN_OPMOD_LOOP            0x0004   /* Internal loopback mode */

/* CAN Configuration */
#define DST_CAN_CFG_SYNC_RTOD         0x0000   /* Sync on recessive to dom edge*/
#define DST_CAN_CFG_SYNC_BOTH         0x0001   /* Sync on both edges */
#define DST_CAN_CFG_SAMPL_0           0x0000   /* Sample mode 0: 1 point */
#define DST_CAN_CFG_SAMPL_1           0x0002   /* Sample mode 1: 3 points */
#define DST_CAN_CFG_SJW_1             0x0000   /* Sync Jump width 1 */
#define DST_CAN_CFG_SJW_2             0x0004   /* Sync Jump width 2 */
#define DST_CAN_CFG_SJW_3             0x0008   /* Sync Jump width 3 */
#define DST_CAN_CFG_SJW_4             0x000C   /* Sync Jump width 4 */
#define DST_CAN_CFG_AUTO_RES          0x0010   /* Auto restart */
#define DST_CAN_CFG_OVR_MSG           0x8000   /* Overwrite last message */

/* CAN acceptance filter enable */
#define DST_CAN_FILT_EN_0                     0x0001   /* Enable filter 0 */
#define DST_CAN_FILT_EN_1                     0x0002   /* Enable filter 1 */
#define DST_CAN_FILT_EN_2                     0x0004   /* Enable filter 2 */

/* CAN arbitration lost capture */
#define DST_CAN_ALCR_FRAME_MSK        0x1F00   /* Frame reference field mask */
#define DST_CAN_ALCR_BIT_MSK          0x003F   /* Mask bit vector */

/* CAN Error capture */
#define DST_CAN_ECR_BIT_MSK           0x003F   /* Mask bit vector */
#define DST_CAN_ECR_RXMOD             0x0040   /* Receiving data */
#define DST_CAN_ECR_TXMOD             0x0080   /* Transmitting data */
#define DST_CAN_ECR_FRAME_MSK         0x1F00   /* Frame reference field mask */
#define DST_CAN_ECR_ERRCOD_MSK        0xE000   /* Error code field mask */

/* CAN frame reference */
#define DST_CAN_FRR_BIT_MSK           0x003F   /* Mask bit vector */
#define DST_CAN_FRR_TXMOD             0x0040   /* Transmitting data */
#define DST_CAN_FRR_RXMOD             0x0080   /* Receiving data */
#define DST_CAN_FRR_FRAME_MSK         0x1F00   /* Frame reference field mask */
#define DST_CAN_FRR_TXBIT             0x2000   /* Current bit state on trans line*/
#define DST_CAN_FRR_RXBIT             0x4000   /* Current bit state on rec line */
#define DST_CAN_FRR_STUFIND           0x8000   /* Stuff bit inserted */
```

```
/* Frame reference field definitions - used in ALCR, ECR & FRR */
#define DST_CAN_FRAMREF_STOP          0x0000   /* Stopped */
#define DST_CAN_FRAMREF_SYNC          0x0001   /* Synchronize */
#define DST_CAN_FRAMREF_IFRAME        0x0005   /* Interframe */
#define DST_CAN_FRAMREF_BUSIDL        0x0006   /* Bus Idle */
#define DST_CAN_FRAMREF_SOF           0x0007   /* Start of frame */
#define DST_CAN_FRAMREF_ARB           0x0008   /* Arbitration */
#define DST_CAN_FRAMREF_CTRL          0x0009   /* Control */
#define DST_CAN_FRAMREF_DATA          0x000A   /* Data */
#define DST_CAN_FRAMREF_CRC           0x000B   /* CRC */
#define DST_CAN_FRAMREF_ACK           0x000C   /* Acknowledge */
#define DST_CAN_FRAMREF_EOF           0x000D   /* End of frame */
#define DST_CAN_FRAMREF_ERRF          0x0010   /* Error flag */
#define DST_CAN_FRAMREF_ERRECH        0x0011   /* Error echo */
#define DST_CAN_FRAMREF_ERRDEL        0x0012   /* Error Delay */
#define DST_CAN_FRAMREF_OVRF          0x0018   /* Overload flag */
#define DST_CAN_FRAMREF_OVRECH        0x0019   /* Overload echo */
#define DST_CAN_FRAMREF_OVRDEL        0x001A   /* Overload delay */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** I2C Register Constants
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/* Control Register */
#define DST_I2C_CNTR_IEN              0x80           /* Interrupt Enable */
#define DST_I2C_CNTR_ENAB             0x40           /* I2C Enable */
#define DST_I2C_CNTR_STA              0x20           /* Send start condition */
#define DST_I2C_CNTR_STP              0x10           /* Send stop condition */
#define DST_I2C_CNTR_IFLG             0x08           /* Interrupt flag */
#define DST_I2C_CNTR_AAK              0x04           /* Send acknowledge */

/*
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
** USB Register Constants
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/* USB Interrupt Status */
#define DST_USB_INT_STAT_STALL        0x80   /* Transaction completed or stalled */
#define DST_USB_INT_STAT_ATTACH       0x40   /* USB peripheral attached */
#define DST_USB_INT_STAT_RESUME       0x20   /* */
#define DST_USB_INT_STAT_SLEEP        0x10   /* USB bus idle for 3 ms */
#define DST_USB_INT_STAT_TOK_DNE      0x08   /* Current token processing done */
#define DST_USB_INT_STAT_SOF_TOK      0x04   /* Start of frame token received */
#define DST_USB_INT_STAT_ERROR        0x02   /* Error - see ERR_STAT register */
#define DST_USB_INT_STAT_USB_RST      0x01   /* USB reset decoded */

/* USB Interrupt Enable */
#define DST_USB_INT_ENB_STALL         0x80   /* Enable stall interrupt */
#define DST_USB_INT_ENB_ATTACH        0x40   /* Enable attach interrupt */
#define DST_USB_INT_ENB_RESUME        0x20   /* Enable resume interrupt */
#define DST_USB_INT_ENB_SLEEP         0x10   /* Enable sleep interrupt */
#define DST_USB_INT_ENB_TOK_DNE       0x08   /* Enable token done interrupt */
#define DST_USB_INT_ENB_SOF_TOK       0x04   /* Enable start of frame interrupt */
#define DST_USB_INT_ENB_ERROR 0x02    /* Enable error interrupt */
#define DST_USB_INT_ENB_USB_RST       0x01   /* Enable reset interrupt */
```

```
/* USB Error Interrupt Status */
#define DST_USB_ERR_STAT_BTS_ERR      0x80    /* Bit stuff error */

        /* Bit 6 is reserved */
#define DST_USB_ERR_STAT_DMA_ERR      0x20    /* DMA error */
#define DST_USB_ERR_STAT_BTO_ERR      0x10    /* Bus turn around timeout */
#define DST_USB_ERR_STAT_DFN8 0x08    /* Data field not 8 bits */
#define DST_USB_ERR_STAT_CRC16        0x04    /* CRC16 error */
#define DST_USB_ERR_STAT_CRC5 0x02    /* CRC5 error (host mode only) */
#define DST_USB_ERR_STAT_EOF          0x02    /* EOF error (peripheral mode only) */
#define DST_USB_ERR_STAT_PID_ERR      0x01    /* PID check failed */

/* USB Error Interrupt Enable */
#define DST_USB_ERR_ENB_BTS_ERR       0x80    /* Enable int on BTS error */
/* Bit 6 is reserved */
#define DST_USB_ERR_ENB_DMA_ERR       0x20    /* Enable int on DMA error */
#define DST_USB_ERR_ENB_BTO_ERR       0x10    /* Enable int on Bus timeout */
#define DST_USB_ERR_ENB_DFN8          0x08    /* Enable int on data not 8 bits */
#define DST_USB_ERR_ENB_CRC16 0x04    /* Enable int on CRC16 failure */
#define DST_USB_ERR_ENB_CRC5          0x02    /* Enable int on CRC5 error */
#define DST_USB_ERR_ENB_EOF           0x02    /* Enable int on EOF error */
#define DST_USB_ERR_ENB_PID_ERR       0x01    /* Enable int on PID check error */

/* USB Status */
#define DST_USB_STAT_ENDP_MSK 0xF0    /* Mask the endpoint bits */
#define DST_USB_STAT_ENDP_TX          0x08    /* Buf desc table updated by TX */
#define DST_USB_STAT_ENDP_ODD 0x04    /* Buf desc updated in odd bank */

/* USB Control */
#define DST_USB_CTL_JSTATE            0x80    /* Live differential receiver */
#define DST_USB_CTL_SE0                       0x40    /* Single ended zero */
#define DST_USB_CTL_TXD_SUSPEND       0x20    /* (Target) packet TX/RX disabled */
#define DST_USB_CTL_TOKEN_BUSY        0x20    /* (Host) executing USB token */
#define DST_USB_CTL_RESET             0x10    /* Generate USB reset */
#define DST_USB_CTL_HOST_MODE_EN      0x08    /* Enable host mode */
#define DST_USB_CTL_RESUME            0x04    /* Resume signalling */
#define DST_USB_CTL_ODD_RST           0x02    /* Reset all BDT to even bank */
#define DST_USB_CTL_USB_EN            0x01    /* Enable USB */

/* USB Address */
#define DST_USB_ADDR_LS_EN            0x80    /* Low speed enable */
#define DST_USB_ADDR_MASK             0x7F    /* Address in low 7 bits */

/* USB Token */
#define DST_USB_TOKEN_PID_OUT 0x10    /* Perform out transaction */
#define DST_USB_TOKEN_PID_IN          0x90    /* Perform in transaction */
#define DST_USB_TOKEN_PID_SETUP       0xD0    /* Perform setup */

/* USB Endpoint control */
#define DST_USB_EP_CTL_HOST_WO_HUB    0x80    /* Enable low speed w/o hub */
#define DST_USB_EP_CTL_RETRY_DIS      0x40    /* Do not retry NAK'd transact */

        /* Bit 5 is reserved */
#define DST_USB_EP_CTL_EP_CTL_DIS     0x10    /* Disable control transfers */
#define DST_USB_EP_CTL_EP_RX_EN       0x08    /* Enable RX transfers */
#define DST_USB_EP_CTL_EP_TX_EN       0x04    /* enable TX transfers */
#define DST_USB_EP_CTL_EP_STALL       0x02    /* The endpoint is stalled */
#define DST_USB_EP_CTL_EP_HSHK        0x01    /* Perform handshaking */

#endif

   waitstates.c - Wait state generation for the DSTni-EX

06-May-2003    WD          Author, creation. Revision 1.00

#include <dos.h>
#include "DstTypes.h"
#include <math.h>
```

```
/*
** This table maps the 16 possible wait state values to the correct bit pattern ** for the wait state
portion of the DSTni-EX chip select registers.
**
** Note that bit position 2 is the "Ignore Ready" bit, which
** is left cleared in this table.
*/
static U16 mau16WaitBits[16] = {
   0x0018, 0x0019, 0x001A, 0x001B,    /*  0,  1,  2,  3 */
   0x0010, 0x0011, 0x0012, 0x0013,    /*  4,  5,  6,  7 */
   0x0008, 0x0009, 0x000A, 0x000B,    /*  8,  9, 10, 11 */
   0x0000, 0x0001, 0x0002, 0x0003     /* 12, 13, 14, 15 */
};
/*
*************************************************************
* FUNCTION:    u16CalcWaitStates(U32 u32CpuSpeed, U16 u16DeviceNs)
*
* ARGUMENTS:   2
*                      u32CpuSpeed - clock speed of the CPU in units of Hertz
*                      u16DeviceNs - device access time in units of nanoseconds
*
* RETURNS:     U16
*                      The number of wait states required for the correct operation
*                      of the memory device.
*
* DESCRIPTION:
*       This function computes the number of wait states required for a memory
*       device, given the clock speed of the CPU and the read-access time of the
*       device. The returned value will be in the range of 0 to 15.
*
*************************************************************
*/
U16 u16CalcWaitStates(U32 u32CpuSpeed, U16 u16DeviceNs)
{
       float   fWaits;

   fWaits = ((float)u16DeviceNs / 1000000000.0) * (float)u32CpuSpeed;

   return (U16)ceil(fWaits);
}
/*
*************************************************************
* FUNCTION:    u16WaitStateBits(U16 u16WaitStates)
*
* ARGUMENTS:   1
*                      u16WaitStates
*
* RETURNS:     U16
*                      The bit pattern for the selected number of wait states.
*
* DESCRIPTION:
*       This function, given the desired number of wait states, will perform a
*       lookup in the wait state bit table and return the wait state bit pattern
*       corresponding to the value.
*
*************************************************************
*/
U16 u16WaitStateBits(U16 u16WaitStates)
{
       U16    u16RetVal;

       if (u16WaitStates <= 15)
              u16RetVal = mau16WaitBits[u16WaitStates];
   else
       u16RetVal = mau16WaitBits[15];

   /* Force the "Ignore Ready" bit on */
   return u16RetVal | 0x0004;
}
```

```
/*
**************************************************************
* FUNCTION:    u16GetWaitStateBits(U32 u32CpuSpeed, U16 u16DeviceNs)
*
* ARGUMENTS:   2
*                      u32CpuSpeed - clock speed of the CPU in units of Hertz
*                      u16DeviceNs - device access time in units of nanoseconds
*
* RETURNS:     U16
*                      The bit pattern for the wait states for the given combination
*                      of CPU Speed and device access time.
*
* DESCRIPTION:
*       This function, given a CPU speed and device access time, will compute
*       the required wait states for that device and return the appropriate wait
*       state bit pattern.
*
**************************************************************
*/
U16 u16GetWaitStateBits(U32 u32CpuSpeed, U16 u16DeviceNs)
{
        return u16WaitStateBits(u16CalcWaitStates(u32CpuSpeed, u16DeviceNs));
}
/*
**************************************************************
* FUNCTION:    vSetChipSelect(U16 u16Reg, U16 u16Val, U32 u32CpuSpeed, *U16 u16DeviceNs)
*
* ARGUMENTS:   4
*                      u16Reg - The I/O address of the register to be programmed
*                      u16Val - The value, other than wait states and ready bit,
*                                     to write to the register.
*                      u32CpuSpeed - The speed of the CPU in units of Hertz
*                      u16DeviceNs - device access time in units of nanoseconds
*
* RETURNS:     None
*
* DESCRIPTION:
*       This function will program a chip select register after setting the
*       appropriate wait state bits. The caller of the function provides the bit
*       values for the upper 11 bits of the chip select register.
*
**************************************************************
*/
void vSetChipSelect(U16 u16Reg, U16 u16Val, U32 u32CpuSpeed, U16 u16DeviceNs)
{
        /* Initially make sure the chip select bits are all zero */
        u16Val &= 0xFFE0;

        /* Set the wait state bits per CPU speed and device speed */
        u16Val |= u16GetWaitStateBits(u32CpuSpeed, u16DeviceNs);

        /* Write to the chip select register */
   outport(u16Reg, u16Val);
}
```

checksum.c This source file contains sample code that demonstrates the use of the DSTni-EX hardware checksum adder.

```
*************************************************************
* FUNCTION:    U16 u16CalcChecksum(U16 *pu16Data, U16 u16Len)
*
* ARGUMENTS:   2
*                      pu16Data - pointer to the buffer of data for which the
*                              checksum is to be computed.
*                      u16Len - Number of 16-bit words to add
*
* RETURNS:     U16 containing the checksum
*
* DESCRIPTION:
*     This function perform a byte-swapped checksum over a range of data,
*      using the hardware checksum generator of the DSTni-EX CPU.
*
*      Limitations: This is designed to perform computation over an even
*              number of bytes. The buffer is expected to start on an even-byte
*              boundary. Extra logic would be required to adjust for odd-byte
*              length or odd-byte boundaries.
*
*              Because this function uses a hardware resource, it must be
*              protected from reentrancy or corrupted sums can result.
*
*************************************************************
*/
U16 u16CalcChecksum(U16 *pu16Data, U16 u16Len)
{
   /* Initialize the checksum adder - make sure it's zeroed */
        outport(DST_REG_CAR, 0);
        outport(DST_REG_CDR, 0);

   while (u16Len)
   {
        outport(DST_REG_CAR, *pu16Data++);
      u16Len--;
   }

   /* Get the result */
        return inport(DST_REG_CDR);
}
```

timer.c This source file demonstrates the use of a two-stage timer. This may be needed when the DSTni-EX is set to run at a high clock speed and a slow timer tick is needed. For example, if you want a timer tick to occur 100 times per second (10ms per tick) and the CPU is set to run at 48 MHz, a timer divisor of 120,000 would be required. Because the timer only supports 16 bits, the value cannot be programmed. In this case, set up Timer 2 to act as a pre-scale to one of the other timers.

```
#define TMR_TICKS_PER_SEC     200           /* 5ms per tick */
static void interrupt vTmrInt(void);
static U32 mu32SysTime;    /* Number of ticks since reset */

/*
*************************************************************
* FUNCTION:    vDstTimerInit(U32 u32CpuSpeed)
*
* ARGUMENTS:   1
*                      u32CpuSpeed - CPU clock speed, in units of Hertz
*
* RETURNS:     None
*
* DESCRIPTION:
*     Initializes Timer and starts it ticking. Timer 2 is set to provide a constant
*              1000 Hz clock to Timer 1. The divisor for Timer 2 is computed
*              based on the given CPU speed.
*
*************************************************************
```

```
*/
void vDstTimerInit(U32 u32CpuSpeed)
{
   U16 far  *pu16Vect;
   U16         u16T2Div;
   U16         u16T1Div;

   /* Save current interrupt state and block interrupts */
   asm pushf
   asm cli

   /* Clear the tick counter */
   mu32SysTime = 0;

   /* Make sure timer is stopped by clearing the EN bit */
   outport(DST_REG_T2CON, 0x4000);
   outport(DST_REG_T1CON, 0x4000);

   /* Clear the current counter */
   outport(DST_REG_T2CNT, 0);
   outport(DST_REG_T1CNT, 0);

   /*
   ** Set the comparators
   ** We want Timer 2 to feed Timer 1 a 1000 Hz clock
   ** We want Timer 1 to generate an interrupt 200 times per second
   ** (That's 5ms per tick)
   */
   u16T2Div = (U16)((u32CpuSpeed / 4UL) / 1000UL);
   u16T1Div = 1000 / (U16)TMR_TICKS_PER_SEC;

   /* Set the timer divisors */
   outport(DST_REG_T2CMPA, u16T2Div);
   outport(DST_REG_T1CMPA, u16T1Div);

   /* Set up the timer interrupt handler */
   pu16Vect = (U16 *)MK_FP(0, DST_IVECT_TMR1 * 4);
   pu16Vect[0] = FP_OFF(vTmrInt);
   pu16Vect[1] = _CS;

   /* Start timer 1 with timer 2 as prescale */
   outport(DST_REG_T1CON, 0xE009);

   /* Start timer 2 in continuous mode */
   outport(DST_REG_T2CON, 0xC001);

   /* Unmask the timer interrupt */
   outport(DST_REG_IMASK, inport(DST_REG_IMASK) & ~DST_IMASK_TMR);

   /* Restore interrupt state */
   asm popf
}

/*
**************************************************************
* FUNCTION:    void vTmrInt( void )
*
* ARGUMENTS:   0
*
* RETURNS:     none
*
* DESCRIPTION:
*     This is the interrupt handler for the timer. It will check the
*     software timers for watchdog, serial port and LEDs
*
**************************************************************
```

```
*/
static void interrupt vTmrInt(void)
{
   /* Clear the Max Count bit */
   outport(DST_REG_T1CON, 0x2009);

   /* Bump the system clock */
   mu32SysTime++;

   /* Clear the interrupt */
   outport(DST_REG_EOI, DST_IVECT_TMR1);
}

/*
**************************************************************
* FUNCTION:    void vDstTimerHalt( void )
*
* ARGUMENTS:   0
*
* RETURNS:     void
*
* DESCRIPTION:
*     Halts operation of the timer
*
**************************************************************
*/
void vDstTimerHalt(void)
{
   /* Make sure timers are stopped by clearing the EN bit */
   outport(DST_REG_T1CON, 0x4000);
   outport(DST_REG_T2CON, 0x4000);

   /* Mask the timer interrupt */
   outport(DST_REG_IMASK, inport(DST_REG_IMASK) | DST_IMASK_TMR);

   /* Clear any timer interrupts that may be pending */
   outport(DST_REG_EOI, DST_IVECT_TMR1);
}

/*
**************************************************************
* FUNCTION:    U32 u32DstGetTime( void )
*
* ARGUMENTS:   0
*
* RETURNS:     U32 Number of elapsed ticks
*
* DESCRIPTION:
*     Returns number of clock ticks since the last reset.
*
**************************************************************
*/
U32 u32DstGetTime(void)
{
   U32   u32Time;

   ** Since this is a 16 bit CPU, we need to protect against the long word SysTime being
   ** changed in the middle of a copy. That's why we disable interrupts during the copy.
   asm pushf
   asm cli
   u32Time = mu32SysTime;
   asm popf
   return u32Time;
               }
```

9: Baud Rate Calculations

This appendix shows the baud rate calculations for the following CPU clock speeds:

- 20 MHz — see page 107
- 24 MHz — see page 108
- 25 MHz — see page 109
- 36 MHz — see page 109
- 48 MHz — see page 110
- 60 MHz — see page 110
- 72 MHz — see page 111
- 84 MHz — see page 111
- 96 MHz — see page 112

Using this information, you can calculate valid baud rate values for the CPU clock speed(s) you want to use. The acceptable error rate is +3% or –2.5%.

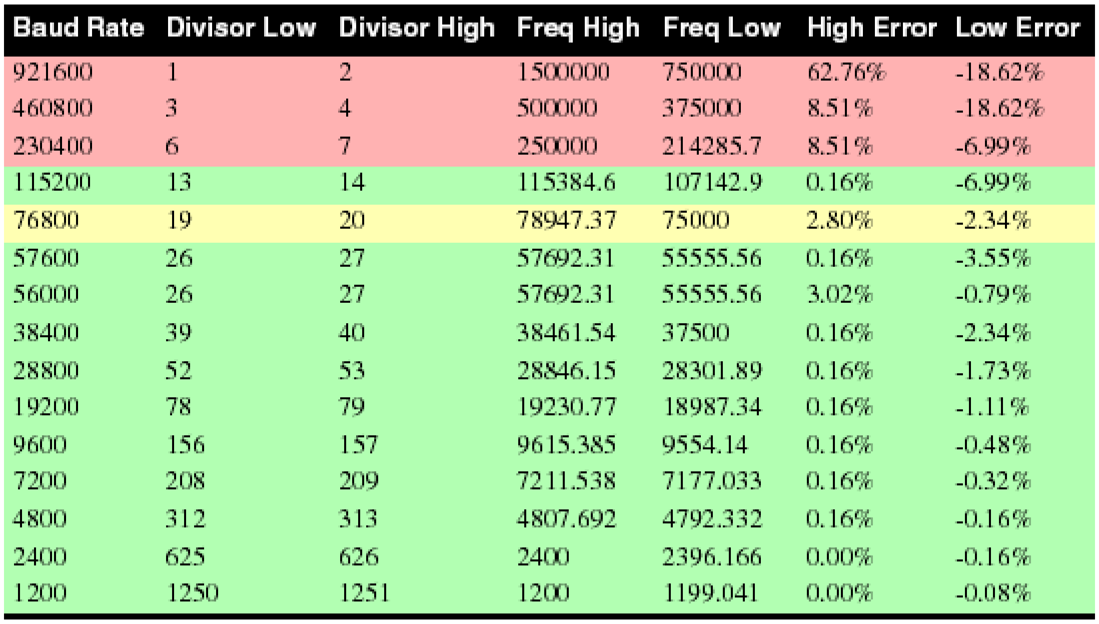
Legend:

Fail	Marginal	Good

Table 9-1. Baud Rate Calculations Using a CPU Clock Speed of 20 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	1	2	1250000	625000	35.63%	-32.18%
460800	2	3	625000	416666.7	35.63%	-9.58%
230400	5	6	250000	208333.3	8.51%	-9.58%
115200	10	11	125000	113636.4	8.51%	-1.36%
76800	16	17	78125	73529.41	1.73%	-4.26%
57600	21	22	59523.81	56818.18	3.34%	-1.36%
56000	22	23	56818.18	54347.83	1.46%	-2.95%
38400	32	33	39062.5	37878.79	1.73%	-1.36%
28800	43	44	29069.77	28409.09	0.94%	-1.36%
19200	65	66	19230.77	18939.39	0.16%	-1.36%
9600	130	131	9615.385	9541.984	0.16%	-0.60%
7200	173	174	7225.434	7183.908	0.35%	-0.22%
4800	260	261	4807.692	4789.272	0.16%	-0.22%
2400	520	521	2403.846	2399.232	0.16%	-0.03%
1200	1041	1042	1200.768	1199.616	0.06%	-0.03%

Table 9-2. Baud Rate Calculations Using a CPU Clock Speed of 24 MHz



Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	1	2	1500000	750000	62.76%	-18.62%
460800	3	4	500000	375000	8.51%	-18.62%
230400	6	7	250000	214285.7	8.51%	-6.99%
115200	13	14	115384.6	107142.9	0.16%	-6.99%
76800	19	20	78947.37	75000	2.80%	-2.34%
57600	26	27	57692.31	55555.56	0.16%	-3.55%
56000	26	27	57692.31	55555.56	3.02%	-0.79%
38400	39	40	38461.54	37500	0.16%	-2.34%
28800	52	53	28846.15	28301.89	0.16%	-1.73%
19200	78	79	19230.77	18987.34	0.16%	-1.11%
9600	156	157	9615.385	9554.14	0.16%	-0.48%
7200	208	209	7211.538	7177.033	0.16%	-0.32%
4800	312	313	4807.692	4792.332	0.16%	-0.16%
2400	625	626	2400	2396.166	0.00%	-0.16%
1200	1250	1251	1200	1199.041	0.00%	-0.08%

Table 9-3. Baud Rate Calculations Using a CPU Clock Speed of 25 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	1	2	1562500	781250	69.54%	-15.23%
460800	3	4	520833.3	390625	13.03%	-15.23%
230400	6	7	260416.7	223214.3	13.03%	-3.12%
115200	13	14	120192.3	111607.1	4.33%	-3.12%
76800	20	21	78125	74404.76	1.73%	-3.12%
57600	27	28	57870.37	55803.57	0.47%	-3.12%
56000	27	28	57870.37	55803.57	3.34%	-0.35%
38400	40	41	39062.5	38109.76	1.73%	-0.76%
28800	54	55	28935.19	28409.09	0.47%	-1.36%
19200	81	82	19290.12	19054.88	0.47%	-0.76%
9600	162	163	9645.062	9585.89	0.47%	-0.15%
7200	217	218	7200.461	7167.431	0.01%	-0.45%
4800	325	326	4807.692	4792.945	0.16%	-0.15%
2400	651	652	2400.154	2396.472	0.01%	-0.15%
1200	1302	1303	1200.077	1199.156	0.01%	-0.07%

Table 9-4. Baud Rate Calculations Using a CPU Clock Speed of 36 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	2	3	1125000	750000	22.07%	-18.62%
460800	4	5	562500	450000	22.07%	-2.34%
230400	9	10	250000	225000	8.51%	-2.34%
115200	19	20	118421.1	112500	2.80%	-2.34%
76800	29	30	77586.21	75000	1.02%	-2.34%
57600	39	40	57692.31	56250	0.16%	-2.34%
56000	40	41	56250	54878.05	0.45%	-2.00%
38400	58	59	38793.1	38135.59	1.02%	-0.69%
28800	78	79	28846.15	28481.01	0.16%	-1.11%
19200	117	118	19230.77	19067.8	0.16%	-0.69%
9600	234	235	9615.385	9574.468	0.16%	-0.27%
7200	312	313	7211.538	7188.498	0.16%	-0.16%
4800	469	469	4807.692	4797.441	0.16%	-0.05%
2400	937	938	2401.281	2398.721	0.05%	-0.05%
1200	1875	1876	1200	1199.36	0.00%	-0.05%

Table 9-5. Baud Rate Calculations Using a CPU Clock Speed of 48 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	3	4	1000000	750000	8.51%	-18.62%
460800	6	7	500000	428571.4	8.51%	-6.99%
230400	13	14	230769.2	214285.7	0.16%	-6.99%
115200	26	27	115384.6	111111.1	0.16%	-3.55%
76800	39	40	76923.08	75000	0.16%	-2.34%
57600	52	53	57692.31	56603.77	0.16%	-1.73%
56000	53	54	56603.77	55555.56	1.08%	-0.79%
38400	78	79	38461.54	37974.68	0.16%	-1.11%
28800	104	105	28846.15	28571.43	0.16%	-0.79%
19200	156	157	19230.77	19108.28	0.16%	-0.48%
9600	312	313	9615.385	9584.665	0.16%	-0.16%
7200	416	417	7211.538	7194.245	0.16%	-0.08%
4800	625	626	4800	4792.332	0.00%	-0.16%
2400	1250	1251	2400	2398.082	0.00%	-0.08%
1200	2500	2501	1200	1199.52	0.00%	-0.04%

Table 9-6. Baud Rate Calculations Using a CPU Clock Speed of 60 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	4	5	937500	750000	1.73%	-18.62%
460800	8	9	468750	416666.7	1.73%	-9.58%
230400	16	17	234375	220588.2	1.73%	-4.26%
115200	32	33	117187.5	113636.4	1.73%	-1.36%
76800	48	49	78125	76530.61	1.73%	-0.35%
57600	65	66	57692.31	56818.18	0.16%	-1.36%
56000	66	67	56818.18	55970.15	1.46%	-0.05%
38400	97	98	38659.79	38265.31	0.68%	-0.35%
28800	130	131	28846.15	28625.95	0.16%	-0.60%
19200	195	196	19230.77	19132.65	0.16%	-0.35%
9600	390	391	9615.385	9590.793	0.16%	-0.10%
7200	520	521	7211.538	7197.697	0.16%	-0.03%
4800	781	782	4801.536	4795.396	0.03%	-0.10%
2400	1562	1563	2400.768	2399.232	0.03%	-0.03%
1200	3125	3126	1200	1199.616	0.00%	-0.03%

Table 9-7. Baud Rate Calculations Using a CPU Clock Speed of 72 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	4	5	1125000	900000	22.07%	-2.34%
460800	9	10	500000	450000	8.51%	-2.34%
230400	19	20	236842.1	225000	2.80%	-2.34%
115200	39	40	115384.6	112500	0.16%	-2.34%
76800	58	59	77586.21	76271.19	1.02%	-0.69%
57600	78	79	57692.31	56962.03	0.16%	-1.11%
56000	80	81	56250	55555.56	0.45%	-0.79%
38400	117	118	38461.54	38135.59	0.16%	-0.69%
28800	156	157	28846.15	28662.42	0.16%	-0.48%
19200	234	235	19230.77	19148.94	0.16%	-0.27%
9600	468	469	9615.385	9594.883	0.16%	-0.05%
7200	625	626	7200	7188.498	0.00%	-0.16%
4800	937	938	4802.561	4797.441	0.05%	-0.05%
2400	1875	1876	2400	2398.721	0.00%	-0.05%
1200	3750	3751	1200	1199.68	0.00%	-0.03%

Table 9-8. Baud Rate Calculations Using a CPU Clock Speed of 84 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	5	6	1050000	875000	13.93%	-5.06%
460800	11	12	477272.7	437500	3.57%	-5.06%
230400	22	23	238636.4	228260.9	3.57%	-0.93%
115200	45	46	116666.7	114130.4	1.27%	-0.93%
76800	68	69	77205.88	76086.96	0.53%	-0.93%
57600	91	92	57692.31	57065.22	0.16%	-0.93%
56000	93	94	56451.61	55851.06	0.81%	-0.27%
38400	136	137	38602.94	38321.17	0.53%	-0.21%
28800	182	183	28846.15	28688.52	0.16%	-0.39%
19200	273	274	19230.77	19160.58	0.16%	-0.21%
9600	546	547	9615.385	9597.806	0.16%	-0.02%
7200	729	730	7201.646	7191.781	0.02%	-0.11%
4800	1093	1094	4803.294	4798.903	0.07%	-0.02%
2400	2187	2188	2400.549	2399.452	0.02%	-0.02%
1200	4375	4376	1200	1199.726	0.00%	-0.02%

Table 9-9. Baud Rate Calculations Using a CPU Clock Speed of 96 MHz

Baud Rate	Divisor Low	Divisor High	Freq High	Freq Low	High Error	Low Error
921600	6	7	1000000	857142.9	8.51%	-6.99%
460800	13	14	461538.5	428571.4	0.16%	-6.99%
230400	26	27	230769.2	222222.2	0.16%	-3.55%
115200	52	53	115384.6	113207.5	0.16%	-1.73%
76800	78	79	76923.08	75949.37	0.16%	-1.11%
57600	104	105	57692.31	57142.86	0.16%	-0.79%
56000	107	108	56074.77	55555.56	0.13%	-0.79%
38400	156	157	38461.54	38216.56	0.16%	-0.48%
28800	208	209	28846.15	28708.13	0.16%	-0.32%
19200	312	313	19230.77	19169.33	0.16%	-0.16%
9600	625	626	9600	9584.665	0.00%	-0.16%
7200	833	834	7202.881	7194.245	0.04%	-0.08%
4800	1250	1251	4800	4796.163	0.00%	-0.08%
2400	2500	2501	2400	2399.04	0.00%	-0.04%
1200	5000	5001	1200	1199.76	0.00%	-0.02%